UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

AN AUTOMATED SYSTEM TO SEARCH, TRACK, CLASSIFY AND REPORT SENSITIVE INFORMATION EXPOSED ON AN INTRANET

Tiago Filipe Eleutério Duarte

Dissertação orientada pela Prof. Doutora Maria Dulce Pedroso Domingos
e co-orientado por José António dos Santos Alegria

DISSERTAÇÃO

MESTRADO EM SEGURANÇA INFORMÁTICA

2015

Acknowledgments

First, I want to thank my family for all the support given during all my (academic) life, specially to my parents Paula and Rui. Their effort was huge, but thankfully, it was all worthed and I hope they are proud of where I am and were I hope to be, so thank you for all the love and support. A special thank goes also to my godparents Zé and Ivone, to my cousin Joana (my "little sister"), and to my grandparents Chico and Graça, for all the support and love they gave and for being such a great pillar in my life. Now to my beautiful girlfriend Margarida, who have always supported me, giving me strength for the most difficult moments and always walking this path by my side, thank you so much for everything, it is because of you that all of this was possible. To my "parents-in-law" Margarida and João, thank you for all your support and advices, as well as for accepting me so well in your family. To my "sisters-in-law" Joana, Pipa and Francisca, as well as my "nephew" Duarte, whom I love, and my "brothers-in-law" Sérgio and Hugo, thank you for all the happy family moments and for all the support, you have been so important for me, as I know you will continue to be. To my great friends Rúben, Bernardo, Andreia, Mica, Claúdio, Inês, Micael and Sam, thank you all for the fantastic moments in high school and even after, and for all the friendship you shared with me, I know that I can always count on you, as much as you can count on me. To my closest friends from FCUL João, Sasha, Sérgio, Ricardo, Pedro, Rita, Mafalda, Marta, Diogo, Juliana, JC and Faneca, thank you for these great years and for the next that will come. To Miguel, Pedro, Bernardo and Pedro, Radu and Rodrigo, mostly for this two late years, thank you for the friendship, the share of knowledge and advising. To all the pilgrims from São Gonçalo, thank you all for allowing me to walk beside you. To all the teachers that had contributed for my learning and knowledge. To my thesis counselors Dulce Domingos, thank you for all the support and help to write this thesis and José Alegria for the support in PT. To my coworkers and mentors Pedro Inácio, Luís Costa (which left PT in the beginning but gave a huge contribution to begin this project) and Paulo Serrão, thank you all for the support and learning. To my thesis colleague Rúben for this last nine months in PT developing our projects, thank you for the co-working.

It is hard to express all the gratitude that I feel towards you, but it is genuine and I am so happy for having this great family and so good friends around me. Life is shown

increasingly difficult, but I know I will get to take the best she has to offer, with happiness and strength that each of you give me. Thank you all for everything, this thesis is for you.

À Margarida, família e amigos

Abstract

Through time, enterprises have been focusing their main attentions towards cyber attacks against their infrastructures derived from the outside and so they end, somehow, underrating the existing dangers on their internal network. This leads to a low importance given to the information available to every employee connected to the internal network, may it be of a sensitive nature and most likely should not be available to everyone's access.

Currently, the detection of documents with sensitive or confidential information unduly exposed on PTP's (Portugal Telecom Portugal) internal network is a rather time consuming manual process.

This project's contribution is Hound, an automated system that searches for documents, exposed to all employees, with possible sensitive content and classifies them according to its degree of sensitivity, generating reports with that gathered information. This system was integrated in a PT project of larger dimensions, in order to provide DCY (Cybersecurity Department) with mechanisms to improve its effectiveness on the vulnerability detection area, in terms of exposure of files/documents with sensitive or confidential information in its internal network.

Keywords: Information Security, Intranet Crawling, Sensitive Information, Information Retrieval, Data Classification

Resumo

Ao longo do tempo, as empresas têm vindo a focar as suas principais atenções para os ataques contra as suas infraestruturas provenientes do exterior acabando por, de certa forma, menosprezar os perigos existentes no interior da sua rede. Isto leva a que não dêem a devida importância à informação que está disponível para todos os funcionários na rede interna, podendo a mesma ser de caráter sensível e que muito provavelmente não deveria estar disponível para o acesso de todos.

Atualmente, a deteção de ficheiros com informação sensível ou confidencial indevidamente expostos na rede interna da PTP (Portugal Telecom Portugal) é um processo manual bastante moroso.

A contribuição deste projeto é o Hound, um sistema automatizado que procura documentos, expostos aos colaboradores, com conteúdo potencialmente sensível. Estes documentos são classificados de acordo com o seu grau de sensibilidade, gerando relatórios com a informação obtida. Este sistema foi integrado num projeto de maiores dimensões da PT de forma a dotar o Departamento de Cibersegurança dos mecanismos necessários a melhorar a sua eficácia nas áreas de deteção de vulnerabilidades, em termos de exposição de ficheiros/documentos com informação sensível ou confidencial na sua rede interna.

Palavras-chave: Segurança de Informação, Prospeção na Intranet, Informação Sensível, Recuperação de Informação, Classificação de Dados

Resumo Alargado

Nas últimas décadas, a tecnologia tem sofrido uma evolução cada vez maior, surgindo sucessivamente novas soluções que são adotadas nas nossas tarefas diárias, sejam elas pessoais ou profissionais. Com esta rápida evolução, surgem também problemas de segurança inerentes e portanto a preocupação com esta vertente tem aumentado. As grandes empresas não são exceção e, dadas as suas dimensões, torna-se difícil o controlo da informação existente na infraestrutura. Este problema deve-se maioritariamente à dimensão das empresas, mas também às questões legais que envolvem o acesso à informação. A Cibersegurança torna-se cada vez mais uma preocupação nas empresas, não só devido a obrigações legais, controlos exigidos, responsabilidade criminal e multas inerentes, mas também pelo impacto que um ataque com intrusão possa ter sobre a imagem da empresa, podendo ter repercussões severas no seu negócio. Apesar de geralmente se atribuir uma grande importância à segurança das infraestruturas, protegendo-as ao máximo de ataques externos, a verdade é que a rede interna de uma empresa é tanto ou mais suscetível a ataques provocados por funcionários, sejam eles deliberados ou acidentais. A dispersão de informação nas empresas é de tal forma elevada, que muita informação considerada sensível ou confidencial, por exemplo dados de clientes ou de acesso a plataformas, poderá estar facilmente acessível a qualquer utilizador com acesso à rede interna da empresa, o que levanta graves problemas.

Posto este problema, surge a importância deste projeto. Dada o difícil controlo sobre a dispersão de informação, torna-se essencial ter um sistema para a pesquisa, classificação e alerta automatizadas, focada na informação sensível exposta de forma desprotegida na Intranet, que percorre a rede sistematicamente à procura de documentos partilhados que, potencialmente ou diretamente, contenham informação da empresa considerada sensível ou confidencial.

O principal objetivo deste projeto é investigar e desenvolver um sistema que automatize ao máximo este processo, que seria demasiado moroso quando efetuado de forma manual. Para tal são procurados documentos de vários tipos (.txt, .doc, .xls, .pdf, etc.) expostos a todos os funcionários em geral em pastas partilhadas e páginas de plataformas web internas, que consiga identificar ficheiros, específicos ou não, que contenham informação potencialmente sensível, gerando alertas e classificando os que são encontrados.

O enquadramento teórico para a realização deste projeto baseia-se em motores de busca, que por sua vez são compostos por *crawlers* (cuja informação foi baseada principalmente em Olston e Najork, 2010 [20] e Batsakis et al, 2009 [2]) e um sistema de *Informatieion Retrieval* (baseado principalmente nas definições de Baeza-Yates e Ribeiro-Neto, 1999 [1], Ceri et al, 2013 [5] e Salton et al, 1975 [23] para o Modelo de Espaço Vetorial). Foram também observadas outras soluções de extração de informação existentes no mercado, sendo elas Datafari, Searchblox e Google Search Appliance, de forma a providenciar uma melhor perceção daquilo que teria de ser feito neste projeto.

A contribuição deste projeto é o sistema Hound, automatizado para a procura de documentos através de pesquisas sucessivas, localizando-os em pastas partilhadas e que os classifica de acordo com o seu grau de sensibilidade, gerando relatórios com um resumo da informação encontrada, caso esta seja de facto sensível. Para tal, foram usadas ferramentas auxiliares para a fase de extração de informação dos repositórios da PT (Windows shares e Sharepoint). A extração é feita pelo *engine* do Hound, com o auxílio da ferramenta Apache ManifoldCF e a indexação da informação obtida é feita através do Elasticsearch. É feito um reconhecimento da rede através do Nmap, de forma a encontrar as máquinas com os portos 139 e/ou 445 abertos. De seguida é feita a configuração automática do Apache ManifoldCF para extrair o conteúdo dos documentos que se encontram nos repositórios encontrados anteriormente e indexar a informação extraida no Elasticsearch, em formato JSON. O módulo do Hound responsável pela procura de informação sensível nos documentos extraídos é o QuerySearchRank, que através de plugins modulares desenvolvidos analiza, com a utilização de expressões regulares ou listas de palavras, as linhas de cada ficheiro à procura de informação sensível (números de telefone, NIF, IPs, emails, passwords, datas, nomes), guardando-a, caso seja encontrada, num novo índice do Elasticsearch. Para a classificação dos documentos, foi criado um modelo, baseado no método sugerido por Mike Chapple [6], de forma a classificar de forma simples e eficiente, os documentos encontrados que contêm informação sensível.

Com a integração do projeto na plataforma da DCY, são fornecidos os mecanismos necessários para melhorar a eficiência da mesma na área de deteção de vulnerabilidades em termos de ficheiros expostos com informação sensível ou confidencial na rede interna e permitindo, graças à sua modularidade e escalabilidade, aumentar o número de plugins de forma a encontrar novos tipos de informação que sejam necessários no futuro.

Palavras-chave: Segurança de Informação, Prospeção na Intranet, Informação Sensível, Recuperação de Informação, Classificação de Dados

Contents

List of Figures

List of Tables

Listings

Abbreviations

AMQP Advanced Message Queuing Protocol.

API Application Programming Interface.

DCY Department of CyberSecurity.

DSL Domain-Specific Language.

DSL Secure Hash Algorithm.

ES Elasticsearch.

HTML HyperText Markup Language.

idf Inverse Document Frequency.

IP Internet Protocol.

IR Information Retrieval.

ISO International Organization for Standardization.

JSON JavaScript Object Notation.

MCF ManifoldCF.

MIME Multi-Purpose Internet Mail Extensions.

NIF Número de Identificação Fiscal.

NISS Número de Identificação de Segurança Social.

PDF Portable Document Format.

PT Portugal Telecom.

PTP Portugal Telecom Portugal.

RTF Rich Text Format.

TCP Transmission Control Protocol.

tf Term Frequency.

TSDB Time Series Database.

URL Uniform Resource Locator.

UTF-8 Universal Character Set + Transformation Format 8-bit.

VAT Value Added Tax.

VSM Vector Space Model.

WI-FI Wireless Fidelity.

WWW World Wide Web.

XML Extensible Markup Language.

Chapter 1

Introduction

"It used to be expensive to make things public and cheap to make them private. Now it's expensive to make things private and cheap to make them public." - Clay Shirky, Internet scholar and professor at N.Y.U.

1.1 Motivation

Nowadays, more than ever, the concern with Cybersecurity on big companies have become a reality. Not only the fact that there are legal obligations, demanded controls, criminal responsibility and inherent fines, but also by the impact that an attack, intrusion and successful illicit access may have on the image and business of a company.

There are some studies which shows that about 80% of the attacks are performed by company employees and only 20% originated by third parties on the Internet [10]. This fact mainly concerns with the dimension of the attack surface, namely, the services, platforms and sensitive information on an intranet that are considerably more exposed and less controlled, increasing the risk of an intrusion or illicit access despite its exposure to an audience far below that of the Internet.

A recent study realizes that, in general, 90% of a security budget is usually spent on protecting network components exposed to Internet and application front ends, and only 10% in other components of distinct areas [19]. However, when there is a security incident in a company, 75% of the time this falls directly on the back-end components that are repositories of information, whatever their nature (DBs, files, shared repositories, etc.), and only 25% of the time is directed to network components [11].

It is precisely in this context that this project gains its significance. Given the scale of the internal network of a large company, the number and variety of devices is huge, so it is normal that the dispersion of information is a difficult reality to control. It becomes essential to have an automated search, crawling, classification and alert system focused on the exposed sensitive information on the Intranet that systematically go through the internal network searching for documents that potentially or directly contain information

of the company considered sensitive.

1.2 Objectives

The objective of this project is to investigate and develop a system that goes through the internal network of PT, seeking documents (e.g. .txt, .doc, .xls, .pdf, .html, etc.) exposed to all employees in shared folders and pages on web-based platforms which, according to its configuration, can identify files, specific or not, that potentially contain sensitive information, generating alerts and ranking the files that are being found.

Taking into account that PT's users can share documents and folders with each other, it is intended with this project to create a system that, automatically, searches for documents that should not be shared with other users, at least unprotected.

1.3 Contributions

The contribution of this project is an automated system that satisfies the proposed objective, which is to search the documents through successive scans, track them in shared folders, classify them according to its sensitivity level and lastly report the information found if it contain, indeed, sensitive information. A classification model was created, adapted from the method defined by Mike Chapple [6], in order to better satisfy this project needs. With its integration in the project cluster of DCY "IntelSources" (Cybersecurity Information Discovery and Intelligence Sources), it provides the necessary mechanisms to improve their effectiveness in the areas of detection of vulnerabilities in terms of file/documents exposure with sensitive or confidential information on the internal network.

1.4 Planning

This project was conducted at PT Comunicações installations at Picoas, Lisboa, starting on October, 1st 2014, with a duration of 9 months. Table 1.1 describes the planning of the project.

Activity	Duration
Study of the crawling technology architecture that will be used	2 weeks
Configuration of tools and crawling mechanisms	1 week
Design and implementation of a simple model	1 week
Definition of the ranking configurations	1 week
Definition of the alert states	1 week
Investigation of information ranking techniques	2 weeks
Elaboration of the preliminary report	1 week
Investigation of machine learning algorithms	2 weeks

Implementation of the machine learning algorithms	2 weeks
Design, architecture and implementation of a learning model in a restricted subnet	4 weeks
Evaluate the scalability and effectiveness of that method on a real and controlled environment	4 weeks
Actualization of the preliminary report	1 week
Application of the model to PT's real internal network	4 weeks
Integration / automation of the developed model implementation	5 weeks
Development of the final report	6 weeks

Table 1.1: Table of project planning

The proposed plan was fulfilled, although there were slight changes over the nine months, which was already expected. The last two points of the plan were mixed according to the stage of labor and due to third-party dependencies to move forward.

1.5 Structure of the document

This document is organized as follows:

- Chapter 2 - Related Work: gives an overview of the information necessary to implement this system, describing a Search Engine, its components (Crawler and Information Retrieval System), Data Searching and Indexation, Data Classification and lastly, similar systems available in the market.
- Chapter 3 - The Hound System: describes the project conducted in the company during the 9 months, explaining in detail each step of the engine developed and how the auxiliary tools were used.
- Chapter 4 - Evaluation: Results of the project's performance and quality of the indexed data.
- Chapter 5 - Conclusion and Future Work: describes the work performed and what could be done in the future to improve the system.

Chapter 2

Related Work

"You can have data without information, but you cannot have information without data."
– Daniel Keys Moran

2.1 Search Engine - Introduction

Search Engines (Figure 2.1) were created to access information around the world and the first one was called Archie, created in 1990 by Alan Emtage. The first crawler-based search engine was WebCrawler, created in 1994. The basis for obtaining information on this project is similar to that used by search engines available on the Internet, like Google, Yahoo, Bing, etc. The main difference remains in the data repository, which is on an Intranet*. Search engines allow users to input keywords that describe the information they need and offer advanced search capabilities.

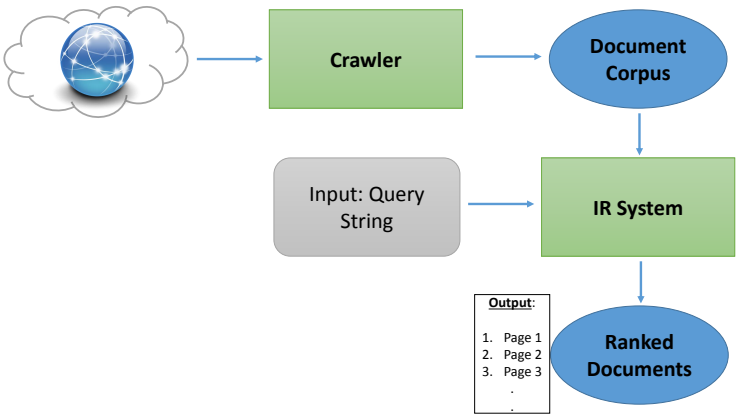

Figure 2.1: Search Engine Architecture

* *Intranet is the generic term for a collection of private computer networks within an organization that uses network technologies as a tool to facilitate communication between people or work groups. The Intranet of an organization typically includes Internet access but it is protected by firewalls and many other security measures, so that its computers cannot be reached directly from the outside.*

The figure shows the architecture of a search engine, where the IR System receives a query and requests the Document Corpus (large and structured set of texts) from the Crawler, that has the task of collecting web-pages. This information is then processed by the IR System, ranking the documents and displaying it to the user.

2.2 Crawler

A Web Crawler (also known as *robot* or *spider*) is a system for the bulk downloading of web pages to build the text collection for the IR system. Web crawlers are used for a variety of purposes. Most prominently, they are one of the main components of web search engines, systems that assemble a corpus of web pages, index them, and allow users to issue queries against the index and find the web pages that match the queries. A related use is web archiving, where large sets of web pages are periodically collected and archived for posterity. A third use is web data mining, where web pages are analyzed for statistical properties, or where data analytics is performed on them. Finally, web monitoring services allow their clients to submit standing queries, or triggers, and they continuously crawl the web and notify clients of pages that match those queries [20].

2.2.1 Design of a Crawler

The design of a crawler (figure 2.2) is basically composed of three main components: a frontier, which stores the list of URLs to visit, a Page Downloader that download pages from WWW, and a Web Repository that receives web pages from a crawler and stores it. In the following, these components are briefly outlined [29].

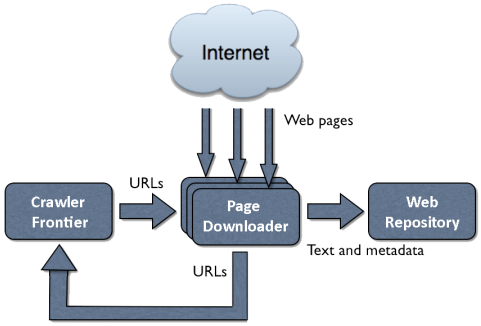

Figure 2.2: Web Crawler Design

1. **Crawler Frontier:** contains the list of unvisited URLs. The working of the crawler starts with the seed URL, where the Crawler retrieves a URL from the frontier. The page corresponding to the URL is fetched from the Web and the unvisited URLs from the page are added to the frontier. The cycle of fetching and extracting the

URL continues until the frontier is empty or some other condition causes it to stop. The extraction of URLs from the frontier is based on some prioritization scheme.

2. **Page Downloader:** downloads the page from the Internet corresponding to the URLs retrieved from the crawler frontier. In order to do that, the Page Downloader requires a HTTP client for sending the HTTP request and to read the response. There should be timeout period defined by the client in order to ensure that it will not take unnecessary time to read large files or wait for response from slow server.

3. **Web Repository:** It is used to store and manage a large pool of web pages. The repository only stores standard HTML pages, ignoring all other media and document types. It is theoretically not that different from other systems that store data objects, such as file systems, database management systems, or information retrieval systems (which is the case for this project). However, a web repository only needs to provide some of the functionalities common to other systems, such as transactions, or a general directory naming structure. It stores the crawled pages as distinct files and the storage manager stores the up-to-date version of every page that the crawler retrieves.

2.2.2 Focused Crawlers

There are many crawler types like the ones used by general purpose search engines, which retrieve massive numbers of web pages regardless their topic, and there are Focused Crawlers that work by combining both the content of the retrieved pages and the link structure of the Web for assigning higher visiting priority to pages with higher probability of being relevant to a given topic. These focused crawlers can be [2]:

- Classic Focused Crawlers: take as input a user query that describes the topic, a set of starting page URLs and guide the search towards pages of interest. Their criteria is assigning priorities to links according to their likelihood to lead to desired pages, being the download order based on the priority defined. These priorities are computed based on the similarity between topic and anchor text of a page or text of page containing the link. This similarity is computed with Boolean or Vector Space Model.

- Semantic Crawlers: variation of Classic Focused Crawlers, but the priorities are assigned by applying semantic criteria for computing page-to-topic relevance: a page and the topic can be relevant if they share conceptually (but not necessarily lexically) similar terms. Conceptual similarity is defined using ontologies.

- Learning Crawlers: apply a training process for assigning visit priorities to web pages and for guiding the crawling process. Higher visit priority is assigned to

links extracted from web pages classified as relevant to the topic and use methods based on *Context Graphs* and *Hidden Markov Models* (HMM).

2.2.3 Incremental Crawlers

A traditional crawler, in order to refresh its collection, periodically replaces the old documents with the newly downloaded documents. On the contrary, an incremental crawler visits them frequently to incrementally refresh the existing collection of pages; based upon the estimate as to how often pages change. It also exchanges less important pages by new and more important pages. It resolves the problem of the freshness of the pages. The benefit of incremental crawler is that only the valuable data is provided to the user, thus network bandwidth is saved and data enrichment is achieved.[29]

2.2.4 Distributed Crawlers

Distributed web crawling is a distributed computing technique. Many crawlers are working to have it in order to get the most coverage of the web. A central server manages the communication and synchronization of the nodes, as it is geographically distributed. It basically uses Page Rank Algorithm for its increased efficiency and quality search. The benefit of distributed web crawler is that it is robust against system crashes and other events, and can be adapted to various crawling applications[29]. An example of a company that uses a distributed crawler architecture is Google [4].

2.3 Information Retrieval

2.3.1 Introduction

Information retrieval (IR) deals with the representation, storage, organization of, and access to information items. The representation and organization of the information items should provide the user with easy access to the information in which he is interested. Unfortunately, characterization of the user information need is not a simple problem. For instance, with this problem "Find all the pages (documents) containing information on college tennis teams which: (1) are maintained by a university in the USA and (2) participates in the NCAA tennis tournament. To be relevant, the page must include information of the national ranking of the team in the last three years and the email or phone number of the team coach". In order to obtain information from this problem, the user must first translate it into a query, which can be processed by the search engine (or IR system). In its most common form, this translation yields a set of keywords (or index terms) which summarizes the description of the user information needs. Given the user query, the key goal of an IR system is to retrieve information which might be useful or relevant to the user. The emphasis is on the retrieval of information as opposed to the retrieval of data [1].

2.3.2 Information vs Data Retrieval

According to Baeza-Yates and Ribeiro-Neto (1999) [1], data retrieval, in the context of an IR system, consists mainly in determining which documents of a collection contain the keywords in the user query which, most frequently, is not enough to satisfy the user information need. In fact, the user of an IR system is more concerned with retrieving information about a subject than with retrieving data which satisfies a given query. A data retrieval language aims at retrieving all objects that clearly satisfy defined conditions such as those in a regular expression or in a relational algebra expression. Thus, for a data retrieval system, a single erroneous object among a thousand retrieved objects means total failure. For an information retrieval system, however, the retrieved objects might be inaccurate and small errors are likely to go unnoticed. The main reason for this difference is that information retrieval usually deals with natural language text, which is not always well structured and could be semantically ambiguous, whereas a data retrieval system deals with data that has a well defined structure and semantics. To be effective in its attempt to satisfy the user information need, the IR system must, somehow, interpret the content of the documents in a collection and rank them according to a degree of relevance to the user query. This interpretation of a document content involves extracting syntactic and semantic information from its text and using it to match the user information need. The difficulty is not only figure how to extract this information, but also knowing how to use it to decide relevance. The primary goal of an IR system is to retrieve all the documents, that are relevant to a user query, while retrieving as few non-relevant documents as possible. Figure 2.3 shows an architecture of the IR Process [5].

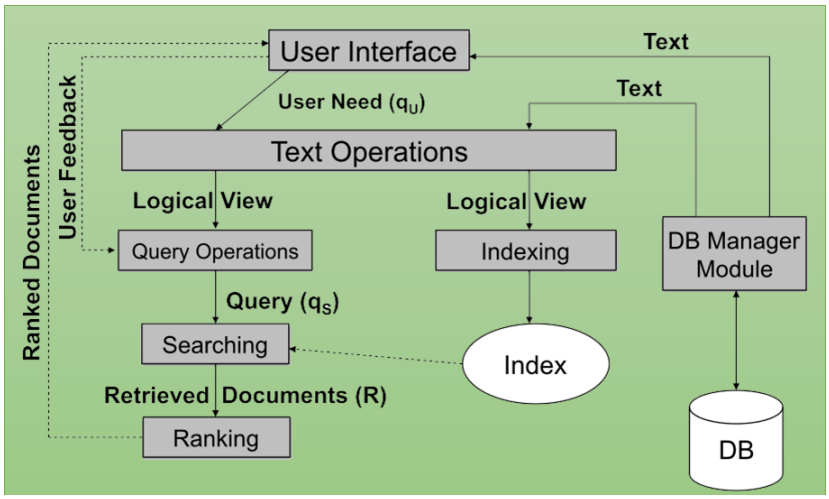

Figure 2.3: Information Retrieval Process

In the figure, it is possible to identify the following subprocesses:

- Database: the source of data of the IR system.

- Indexing: Process that organizes the document retrieval, where Text Operations are applied to transform the documents, generating a logical view of them.
- Query Operations: change the representation of the user need to a logical query.
- Searching: Step of document retrieval according to the interest of the user, obtaining the previously indexed data
- Ranking: Classification of the retrieved documents based on the relevance of the information to the user need.

2.3.3 Information Retrieval Models

There are several IR Models (fig. 2.4), but only the Classic IR Models (Unstructured Text) will be discussed because these are the relevant ones for this project [1].

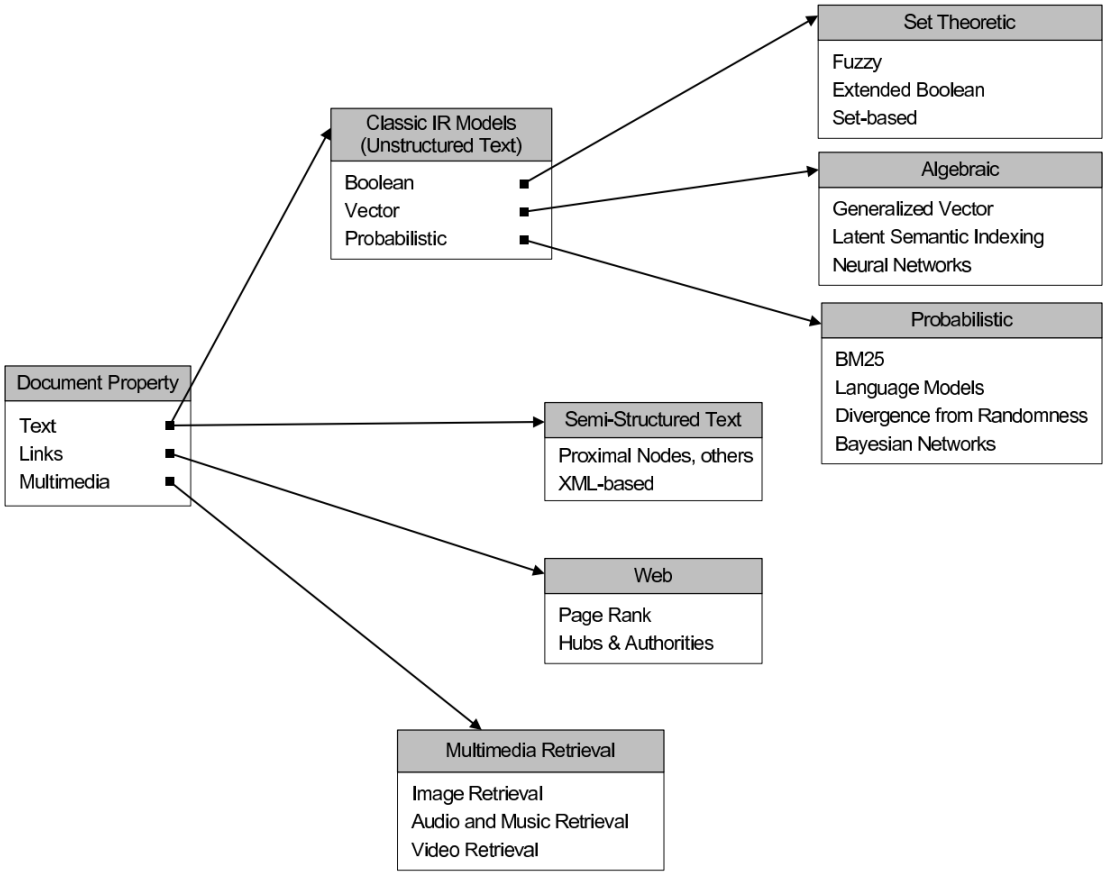

Figure 2.4: Information Retrieval Models

2.3.3.1 Boolean Model

In the Boolean model, a document is represented as a set of keywords, where queries are Boolean expressions of keywords connected by AND, OR, and NOT, including the use of

brackets to indicate the scope of these operators. For example, the query ”all the hotels in Rio Brazil or Hilo Hawaii, but not Hilton” have the following configuration (listing 2.1):

Listing 2.1: Boolean Query

```
[[[Rio & Brazil] | [Hilo & Hawai]] & hotel & !Hilton]
```

The output of the system is a list of documents that are relevant, but there will be no partial matches or ranking. The Boolean model is very rigid, where AND means ”all” and OR means ”any”. All matched documents will be returned, which makes it difficult to control the number of retrieved documents. All the matched documents satisfy the query to the same degree and that makes it difficult to rank the output. Another disadvantage of this model is that it is difficult for the users to express complex queries[14].

Having these disadvantages into account, it became clear that it was not the most adequate model for this project.

2.3.3.2 Vector Space Model

The vector space model procedure, proposed by Salton (1975)[23], can be divided into three stages: Document Indexing, Term Weighting and Similarity Coefficients [27].

I Document Indexing

Many of the words in a document (the, a, is, etc.) do not describe its content, so they are removed from the document vector using an automated indexing, leaving only content bearing words as the representation of the document. This indexing can be based on term frequency, where terms that have both high and low frequency within a document are considered to be function words. In practice, term frequency was difficult to implement in automatic indexing so instead, it is used a stop list which holds common words to remove high frequency words (stop words), which makes the indexing method language dependent. In general, 40-50% of the total number of words in a document are removed with the help of a stop list [27].

II Term Weighting

The term weighting, i.e. its weight/importance, is calculated by multiplying the following three factors together[27]:

- Term frequency: stated by Luhn [**?**], which is the basis of a weighted document vector and corresponds to the number of occurrences of the terms in a document.
- Collection frequency: the general weighting scheme used to discriminate one document from another. The inverse document frequency scheme, assume that the importance of a term is proportional with the number of documents the term appears in.

- Document length normalization: long documents have usually a much larger term set than short documents, which make them more likely to be retrieved.

Different weight schemes have been investigated and the best results, w.r.t. recall and precision, are obtained by using Term Frequency with Inverse Document Frequency and length normalization (TF-IDF)[21]. Basically, this scheme evaluates how important a term is to a specific document in the context of a set of documents (corpus). It is calculated by combining these two metrics (TF and IDF) in a collection of N documents, being TF the measuring of the relevance of a specific document d to a term t (TF_{td}) by calculating the number of occurrences of t in d. Intuitively, the more frequently a term occurs in the document, the more relevant the document is. The term frequency TF_{td} is computed as follows:

$$TF_{td} = \frac{f_{td}}{max_k f_{kd}} \tag{2.1}$$

The IDF for a term is defined as follows: suppose term t appears in n_t of the N documents in the collection. Then:

$$IDF_t = \log_2 \frac{N}{n_t} \tag{2.2}$$

The total TF.IDF score for a term t and a document d is calculated as:

$$TF.IDF_{td} = TF_{td} \times IDF_t \tag{2.3}$$

A simplified approach of applying the weighting to the index list is the construction of a table with the documents listed across the top and the identified content terms listed down the side (figures 2.5 and 2.6), with the weighting being equal to the number of occurrences within the document.

Document 1	Human computer interaction with respect to Java applications
Document 2	A survey of user opinion of computer system response times
Document 3	Magnetism: its effect on computer storage
Document 4	System and human system engineering testing of EPS
Document 5	Relation of user perceived response time to error measurement
Document 6	The generation of random binary ordered trees using Java
Document 7	The intersection graph of paths in trees
Document 8	Graph minors IV: Widths of trees and well quasi ordering
Document 9	Graph minors: A survey

Figure 2.5: Representing Document Space - Documents

Term	Documents								
	1	2	3	4	5	6	7	8	9
human	1			1					
interaction	1								
computer	1	1	1						
Java	1					1			
user		1			1				
system		1		2					
response		1			1				
time		1			1				
magnetism			1						
effect			1						
storage			1						
EPS				1					
survey									1
trees						1	1	1	
graph							1	1	1
minors								1	1

Figure 2.6: Representing Document Space - Table

2.3.3.3 Probabilistic Model

The probabilistic retrieval model [24] is based on the Probability Ranking Principle, which states that an information retrieval system should rank the documents based on their probability of relevance to the query, given all the evidence available [3]. The principle takes into account that there is uncertainty in the representation of the information need and text. There is a variety of sources of evidence used by the probabilistic retrieval methods, being statistical distribution of the terms, in both the relevant and non-relevant texts, the most typical one. Turtle and Croft (1991) [28] developed a system that uses Bayesian inference networks to rank documents, which says that an inference network consists of a directed acyclic dependency graph, where edges represent causal relations or conditional dependency between propositions represented by the nodes (figure 2.7).

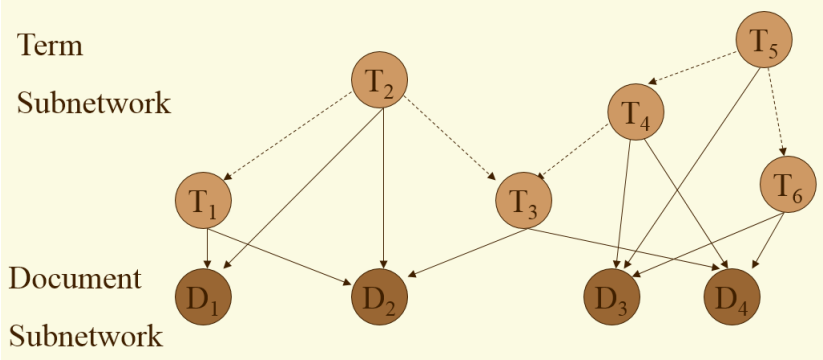

Figure 2.7: Bayesian Network Retrieval Model

The inference network consists of a document network, a concept representation network that represents indexing vocabulary, and a query network representing the informa-

tion need. The concept representation network is the interface between documents and queries. To compute the rank of a document, the inference network is instantiated and the resulting probabilities are propagated through the network to derive a probability associated with the node representing the information need. These probabilities are used to rank documents.

2.3.3.4 Comparing Models

Of all the three models, the Boolean Model is considered to be the weakest, because it does not provide partial matches. Croft [28] suggested that the probabilistic model provides a better retrieval performance, however Salton [23] et al showed that the vector space model outperforms it with general collections.

The statistical approaches, Vector Space (1 & 2) and Probability Models (3), have the following strengths: 1) they provide users with a relevance ranking of the retrieved documents, enabling users to control the output by setting a relevance threshold or by specifying a certain number of documents to display; 2) queries can be easier to formulate because users do not have to learn a query language and can use natural language; 3) the possibility to represent the uncertainty inherent in the choice of query concepts.

However, statistical approaches have some shortcomings: 1) limited expressive power, e.g. problem expressing NOT, because only positive weights are used; 2) statistical approach lacks the structure to express important linguistic features such as phrases; 3) computation of the relevance scores can be computationally intensive; 4) limited view of the information space and it does not directly suggest how to modify a query if the need arises; 5) queries need a large number of words to improve the retrieval performance. As is the case for the Boolean approach, users are faced with the problem of having to choose the appropriate words that are also used in the relevant documents.

This project will follow the Vector Space Model combined with the Boolean Model (not exactly like Extended Boolean Model - even though this is a mix of the two models) to determine the relevance of a given document to a user's query, which is used by Apache Lucene as will be later described. The reason of joining this two models is due to the Vector Space Model difficulty in representing NOT expressions.

2.4 Data indexation and Searching

For data indexation and searching, we focused on Apache Lucene library, because of its high-performance, full-featured text search engine. "Lucene scoring uses a combination of the Vector Space Model (VSM) of Information Retrieval and the Boolean model to determine how relevant a given Document is to a User's query. In general, the idea behind the VSM is the more times a query term appears in a document relative to the number of times the term appears in all the documents in the collection, the more relevant that

document is to the query. It uses the Boolean model to first narrow down the documents that need to be scored based on the use of boolean logic in the Query specification." [17]

Lucene itself is just an indexing and search library and does not contain crawling and HTML parsing functionality. However, several projects extend Lucene's capability like Apache Solr and Elasticserch.

2.4.1 Apache Solr

Apache Solr makes it easy for programmers to develop sophisticated, high-performance search applications with advanced features such as faceting (arranging search results in columns with numerical counts of key terms). Solr is built on top of Lucene, so it inherits its functions and both Solr and Lucene are managed by the Apache Software Foundation.

Solr is featured with Near Real-Time Indexing, Schemaless, Full Text Search with powerful matching capabilities, Restful API - XML, CSV, JSON format, its optimized for High Volume Traffic, Comprehensive Administration Interfaces, Easy Monitoring with metric data via JMX, Highly Scalable and Fault Tolerant because it is built on Apache Zookeeper, Flexible and Adaptable with easy configuration and it has an Extensible Plugin Architecture which makes it easy to connect to other components.

It comes with a front-end platform that allows a much easier interaction and visualization of the data obtained.

2.4.2 Elasticsearch

Elasticsearch is built on top of Lucene, so it inherits its functions. Designed from the ground up for use in distributed environments where reliability and scalability are must haves, Elasticsearch gives the ability to move easily beyond simple full-text search.

Elasticsearch [8] is featured with a Real Time Data and Analytics, it is Distributed with horizontal scaling, Automatic cluster reorganization, High availability with resilient clusters, detection and removal of failed nodes, Multi-Tenancy because one cluster can host multiple indexes queried independently or as a group, Full Text Search with Lucene, Document-Oriented storing data as JSON documents, Conflict Management with optimistic version control to ensure data is never lost due to conflicting changes, nearly Schemaless detecting data structure, indexing data and making it searchable, Restful API with JSON format over HTTP and Per-Operation Persistence where document changes are recorded in transaction logs to minimize data loss.

Elasticsearch data are stored in a cluster with as many nodes as we want [9]. In order to add new nodes, you just need to launch another instance of Elasticsearch with the name of the node in the configuration file and it will be added to the cluster automatically (it has a functionality similar to Zookeeper to manage its nodes). Figure 2.8 shows an example of a cluster with 3 nodes (the node 1 is the master), where each node has 2 shards. With a

total of six shards (three primaries (green) and three replicas (grey)), the index is capable of scaling out to a maximum of six nodes, with one shard on each node and each shard having access to 100% of its node's resources.

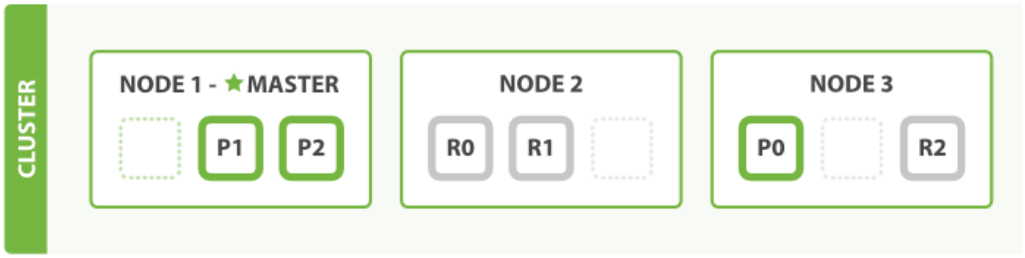

Figure 2.8: A three-node cluster

Elasticsearch data location is defined by an index (similar to a database), a type and an id (identifier of the document) [7]. The UUID (universally unique identifiers) are unique ids for documents and are composed of the type and the id fields, so it's possible to have the same id with different types. This UUIDs can be defined manually (setting a custom id) or automatically (id automatically generated - Auto-generated IDs are 22 characters long, URL-safe, Base64-encoded string) as can be seen in figure 2.9.

```
"_index":    "website",
"_type":     "blog",
"_id":       "123",
"_version":  1,
"created":   true
```

```
"_index":    "website",
"_type":     "blog",
"_id":       "wM0OSFhDQXGZAWDf0-drSA",
"_version":  1,
"created":   true
```

Figure 2.9: Manual id vs Auto Generated id

The good thing in Elasticsearch is that if you only have string type data, you don't need to define a mapping, it will be created automatically. A mapping looks like the one in listing 2.2, where in this example twitter is the index and the mapping refers to the type tweet. It states that the tweet has got a parameter called message of type string and is stored, which means that it will be parsed from the content field when accessed (if false, it would be necessary to parse on the client side).

Listing 2.2: Mapping example

```
$ curl –XPUT 'http://localhost:9200/twitter/_mapping/tweet' –d '
{
  "tweet" : {
    "properties" : {
      "message" : { "type" : "string", "store" : true }
    }
  }
}
```

Having the mapping, the data is indexed as illustrated in listing 2.3. The index is twitter, the type is tweet and the id is 1, with the parameters being user, post_date and message.

Listing 2.3: Document indexing example

```
$ curl -XPUT 'http://localhost:9200/twitter/tweet/1' -d '{
    "user" : "kimchy",
    "post_date" : "2009-11-15T14:12:12",
    "message" : "trying out Elasticsearch"
}'
```

To access the information stored in the indices, we must make a GET command like the one in listing 2.4 - simple GET without query.

Listing 2.4: Simple GET Request

```
$ curl -XGET 'http://localhost:9200/twitter/tweet/1'
```

The result of this query is:

Listing 2.5: Document get result

```
{
  "_index" : "twitter",
  "_type" : "tweet",
  "_id" : "1",
  "version" : 1,
  "found" : true,
  "_source" : {
    "user" : "kimchy",
    "post_date" : "2009-11-15T14:12:12",
    "message" : "trying out Elasticsearch"
  }
}
```

However if we want more specific information from a document, then it's necessary to build a query with the GET command (Boolean, Match, Query String, Filtered, etc.), so that we can get more approximately data to what we wanted to obtain. Listing 2.6 shows an example of a GET request with a query.

Listing 2.6: GET Request with query

```
$ curl -XGET http://localhost:9200/_search -d '{
  "query" : {
    "filtered": {
      "query" : {
        "match: { "tweet": "full text search" }
      }.
      "filter" : {
        "range: { "created": { "gte": "now - 1d / d" }}
      }
}
```

Elasticsearch provides a front-end called Kibana, which allows to see the data in statistic graphics like histograms and geomaps. It contains a payed security module called Shield, that protect data with encrypted communications and authentication.

The best solutions that I have found to achieve the objective of indexing the documents content were Apache Solr and Elasticsearch. I found both solutions quite similar and very efficient, based on [26], so the decision was difficult to make. Elasticsearch has the advantage of being built from the ground to be distributed while Apache Solr needs Zookeeper to do it. Furthermore, Solr has been used for a longer time than Elasticsearch. Even if Elasticsearch is a new player in this game of data, I chose it because of its potential and, as statistics shows, Elasticsearch is gaining interest over time [2.10]. The turning point in interest in these two tools occurred in May 2014 and since then has been constantly growing for Elasticsearch, unlike Solr, which has stagnated.

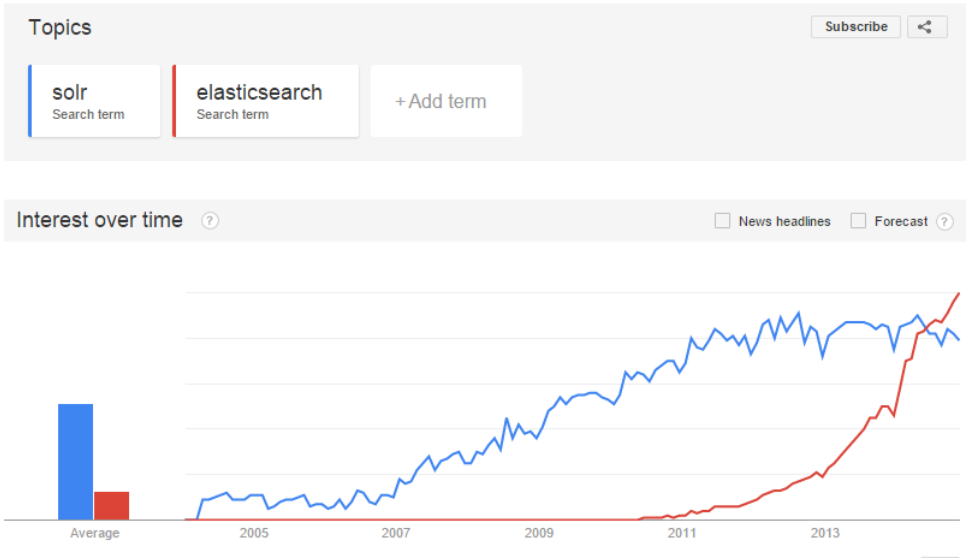

Figure 2.10: Interest in Solr and Elasticsearch since 2004, retrieved from [12].

2.4.3 Apache ManifoldCF

Apache ManifoldCF is an effort to provide an open source framework for connecting source content repositories like Microsoft Sharepoint with indexers like Apache Solr or ElasticSearch. Apache ManifoldCF also defines a security model for target repositories that allows them to enforce source-repository security policies (figure 2.11).

This framework eases the work of extracting the contents of the files, being necessary to configure the connectors in order to define the data repository and the format in which these will be stored after the extraction. This way, I avoid having to retrieve the files, open them, extract its content and store it to be analyzed later, since it do everything by itself.

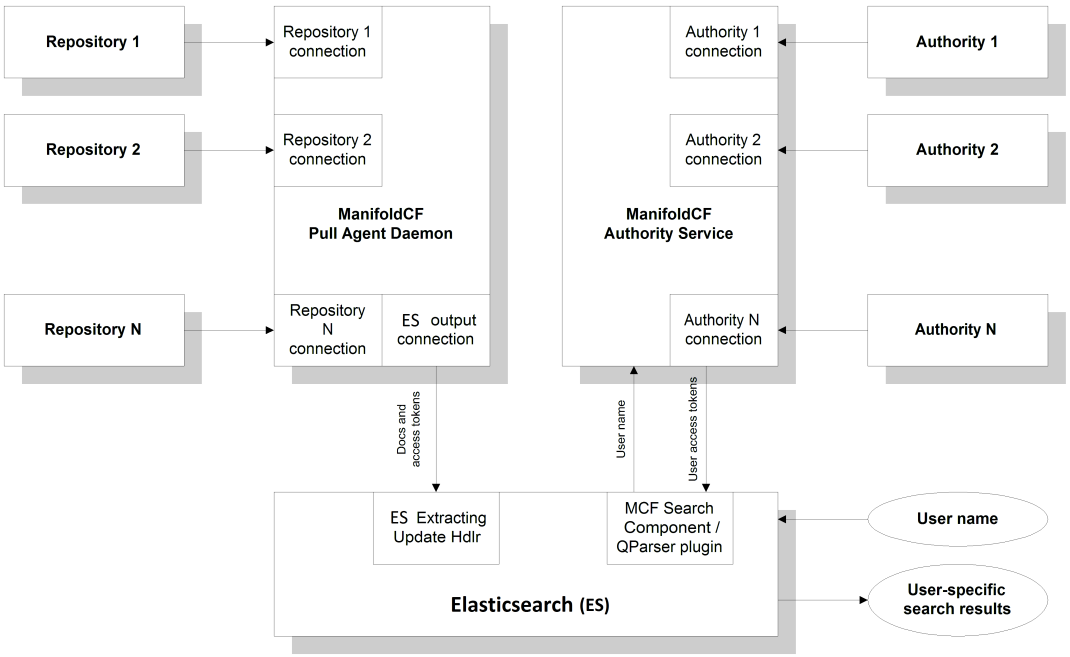

Figure 2.11: ManifoldCF Architecture

Regarding the connectors configuration on ManifoldCF, we have:

- Output Connection: Connector to represent data (Elasticsearch, Solr, HDFS, etc.)
- Transformation Connection: Connector to change the file information (Apache Tika, Metadata Adjuster, etc.)
- Authority Group: Group of the authorities defined on Authority Connections
- Authority Connection: Connector that give access information (Sharepoint/Native ou Active Directory, LDAP, CMIS, etc.)
- Repository Connection: Connector to the repositories where the files are (Sharepoint, Windows Share, HDFS, File System, Google Drive, etc.)
- Job Connection: Connector to run jobs, using a combination of the connections mentioned above.

In order to understand how ManifoldCF works, I resorted to Manifold in action (Wright, Karl D., 2012) [32] in which is explained the operation of connectors supported by the framework. This book is a great assistance for the starters in using Apache ManifoldCF, since there isn't much information on the web and it is the best documentation available.

2.5 Data Classification

The classification of data follows two different visions. Regarding the terms found in each document I used Symantec's top 10 breach type of information [25] to choose the information that I am searching. As for the document classification itself, I adapted Mike Chapel's definition [6]. These approaches will now be detailed.

According to Symantec in its 2015 Internet Security Threat Report [25], "Real names, government ID numbers, and home addresses were the top three types of information breached in 2014. The exposure of financial information grew from 17.8 percent to 35.5 percent in 2014, the largest increase within the top 10 list of information types exposed" (figure 2.12).

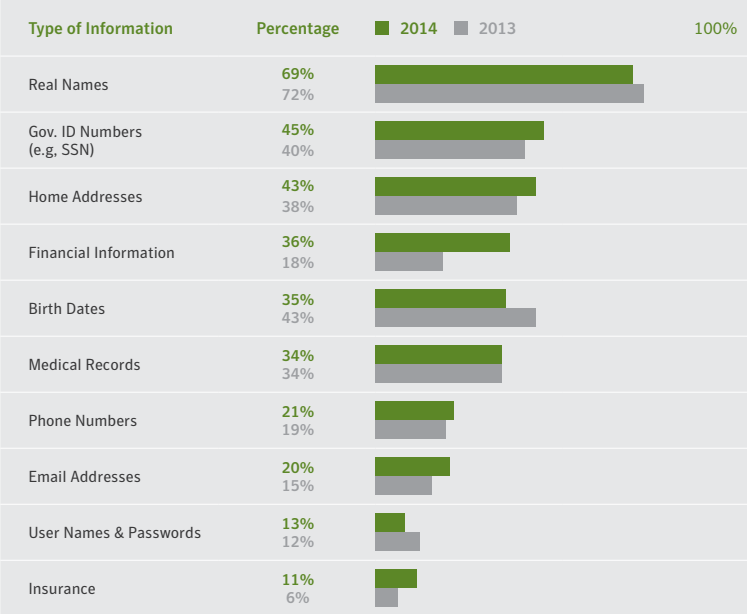

Figure 2.12: Top 10 Types of Information Exposed in 2014

This top 10 list of information exposed was important to define the types of information that would be found in this project. However, it will not be strictly followed, because not all those types of information are important to find and some others are not there referred, so the types of information used are names, phones, email and username/passwords. Also I added marketing plans, IP addresses, VAT numbers and dates (not only birth dates). This changes were made because of its relevance on the company's context.

Regarding the document classification, Mike Chapple [6] says "In my experience, the most critical factor to the success of an information classification program is simplicity. If your program is difficult to understand or the categories are ill-defined, people simply won't use it. The bookshelves of security professionals around the world are littered with binders containing information classification plans that never saw practical implementa-

tion". He defines a four-tier classification model that limits the highest category to a small number of easily recognizable data elements as follows:

- Highly sensitive data - information requiring an extremely high level of oversight and control due to potential reputational, financial or operational impact if improperly disclosed. This should be limited to a clear list of carefully enumerated elements, such as Social Security numbers, credit card numbers and drivers license numbers.

- Sensitive data - information intended for limited use that, if improperly disclosed, could have a serious adverse effect on the organization. This is the "you know it when you see it" category that contains information the organization considers confidential but does not meet the "highly sensitive" bar. For example, this might include plans for the development of new products that have not yet been publicly released.

- Public data - information that may be freely released to the public without concern for confidentiality. This category includes information that you would publish on your website or hand out at a trade show, such as product brochures, public price lists and basic contact information from your firm.

- Internal data - everything else. It is information that you would not freely publish on the Internet, but would not really damage the company if it were accidentally released, such as your internal telephone directory or the ordering list for your supply room. Once you define your data-classification scheme, you need to appropriately classify all of your organization's data, and then develop and implement the security standards that specify appropriate handling practices for each category.

According to OWASP 6 levels of impact (figures 2.13 and 2.14), this project stands in Technical Impacts because the our target is the sensitive data, and in Business Impacts, due to the damage that an exposure of sensitive data can do to the company's reputation.

Threat Agents	Attack Vectors	Security Weakness	
Application Specific	**Exploitability** **DIFFICULT**	**Prevalence** **UNCOMMON**	**Detectability** **AVERAGE**
Consider who can gain access to your sensitive data and any backups of that data. This includes the data at rest, in transit, and even in your customers' browsers. Include both external and internal threats.	Attackers typically don't break crypto directly. They break something else, such as steal keys, do man-in-the-middle attacks, or steal clear text data off the server, while in transit, or from the user's browser.	The most common flaw is simply not encrypting sensitive data. When crypto is employed, weak key generation and management, and weak algorithm usage is common, particularly weak password hashing techniques. Browser weaknesses are very common and easy to detect, but hard to exploit on a large scale. External attackers have difficulty detecting server side flaws due to limited access and they are also usually hard to exploit.	

Figure 2.13: OWASP 6 levels of impact - 1 to 4

Technical Impacts	Business Impacts
Impact SEVERE	**Application / Business Specific**
Failure frequently compromises all data that should have been protected. Typically, this information includes sensitive data such as health records, credentials, personal data, credit cards, etc.	Consider the business value of the lost data and impact to your reputation. What is your legal liability if this data is exposed? Also consider the damage to your reputation.

Figure 2.14: OWASP 6 levels of impact - 5 and 6

2.6 Other Tools

Some tools were consulted in order to understand how they operate, giving a clearer idea of the actions required for implementing this project. If any of these tools were complete enough to satisfy the objectives of this project, they could have been used or even adapted, but that was not the case as we can see next.

2.6.1 Datafari

Datafari is an advanced open source enterprise search solution, modular and reliable (figure 2.15). It can be used as a product for a third party solution.

Figure 2.15: Datafari screen

This solution is similar to the Crawling technology implemented in this project, and it uses Apache Solr 4, Apache ManifoldCF and AjaxFranceLabs (a graphical framework in HTML5/Javascript). It is modular and allows its users to add the connectors they wish to

ManifoldCF and any plugin to Solr, as well as a custom user interface. It is free, since it uses Apache v2 licence and free tools.

However, this solution is not enough to satisfy the proposed objective, since it indexes everything in the target repositories and it uses Solr instead of Elasticsearch (which was the one that I had already decided to use), so it is necessary to look for another solution.

2.6.2 SearchBlox

SearchBlox is an Enterprise Search & Analytics solution built on Apache Lucene & Elasticsearch, that uses Apache ManifoldCF to connect to different repositories and provide authentication (figure 2.16). It gives solutions for Enterprise Search, Text Analytics and Data Sciences.

Figure 2.16: SearchBlox Screen

Of the three solutions, I am interested in the first - Enterprise Search - because of its similarity with this project. It uses Apache ManifoldCF and Elasticsearch for data storage and searching. The graphical framework is a restructure of ManifoldCF web framework. Even though the solution has a free version, in order to use its full features, it is necessary to have a paid version. Also, this solution does not classify the information the way it is necessary for this project.

2.6.3 Google Search Appliance

Google Search Appliance enhances searches with extensive synonyms and intelligent spell-check, just like Google.com, works in more than 20 different languages and auto-completes queries to offer the type of helpful suggestions (figure 2.17).

This solution, unlike the previous ones, is hardware based and is directly applied to a rack. This basically indexes everything that exists in the internal network where it is connected, having several connectors for content extraction from different repositories

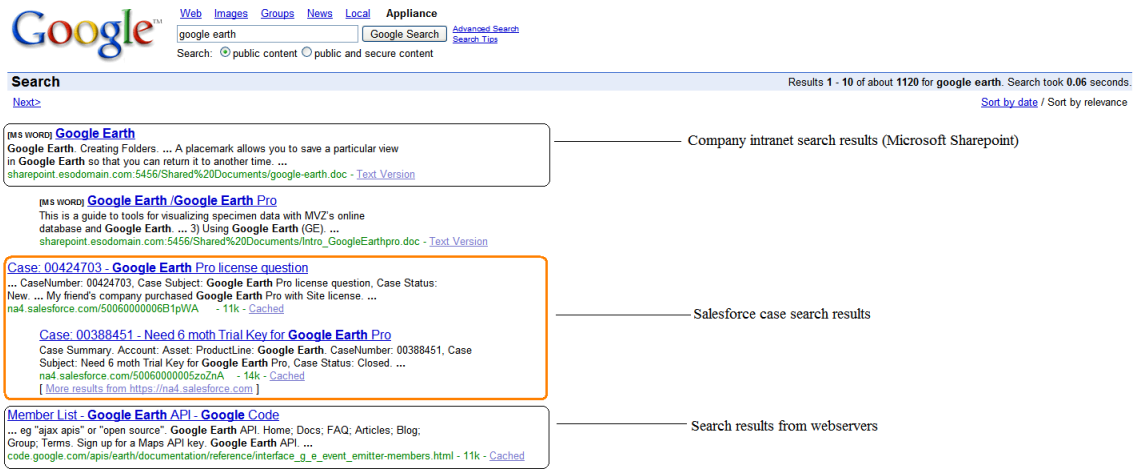

Figure 2.17: Google Search Appliance screen

(Oracle, MySQL, Microsoft Sharepoint, etc.). Since it indexes every data, it poses some privacy problems to the company that uses it for obvious reasons, so it is not the best solution, given the sensitivity of the project. Of the three solutions presented, this one is obviously the most complete and efficient, even though it does not do what is intended for this project.

So, taking into account that I had some time for this project and the needs are very specific, I decided to build the whole engine from the ground, using the original ManifoldCF and Elasticsearch as auxiliary tools.

2.7 Chapter Conclusion

This chapter described the information that was gathered in order to give the necessary background to implement the project. Information Retrieval is in the basis of the operation of the whole project and the chosen IR model was Vector Space with Boolean Model (used by Lucene). Also, some similar solutions that already exist in the web have been referred because during the project, they have been consulted to check if there was an already good and implemented solution that could be adapted for the project. As we have seen in this chapter, these solutions weren't used because they weren't enough to fulfill the purpose of the project, so the whole solution, that uses some of the technologies detailed above, is described in the next chapter thoroughly.

Chapter 3

The Hound System

The aim of this project is to find documents with potentially sensitive information available in open/unprotected repositories to PT's employees. To do so, it is critical that we make a good definition of what is sensitive information in this company's context. This chapter will start by having a Requirements Analysis, followed by the description of the Hound's Architecture and lastly the implementation of the Hound system.

3.1 Requirements Analysis

- Sensitive Information - The types of information that are considered sensitive on the company's context are phone numbers, VATs, emails, IP addresses, dates, names, marketing plans and username/password pairs.
- Scalable - it must grow with the needs of each scan, according to the range of repositories that are indexed, having a good performance for any number of targets.
- Modular - the system should be plugin-based, having a priori defined plugins, and allowing the possibility of easily adding new functionalities, like indexing new types of information.
- Ruby designed - the system's engine should be designed in Ruby, using as many Ruby Gems as necessary, to maximize the compatibility with PT's infrastructure.
- Documents Ranking - have a classification with two possible values: Sensitive and High Sensitive, adapted from Mike Chapple's 2.5 definition. Only these were considered relevant because the others are not relevant, hence they are discarded.
- Level of severity - based on a color system (green, yellow, orange and red), which are calculated according to the ratio of occurrence of the sensitive terms found in the documents and the total number of their lines.
- Centralized - it is important that it can be executed simultaneously in different places at the same time and send the results to the same PT's platform (which is centralized and fault-tolerant), but it is not that relevant to have multiple instances running on the same machine.

- Automatic - The system should be automatic, as the title of this thesis implies, so the user just needs to define the initial parameters.

3.2 Architecture of the Hound* System

After all the analysis and investigation made, I designed a search engine that employs other technologies together to achieve the purposed objective defined above.

Taking into account the search engine in figure 2.1 and the IR model shown in figure 2.3, the Hound System consists of two components visible in figure 3.1:

- ManifoldCF - The Crawler, that obtains the documents from the shares and send the corpus to the IR System, i.e. Hound Engine.
- Hound Engine - The IR System, composed of the Hound Master and the Hound QuerySearchRank, designed in Ruby, doing the text operations, the queries, the search through Elasticsearch and the Ranking using the classification model defined above.

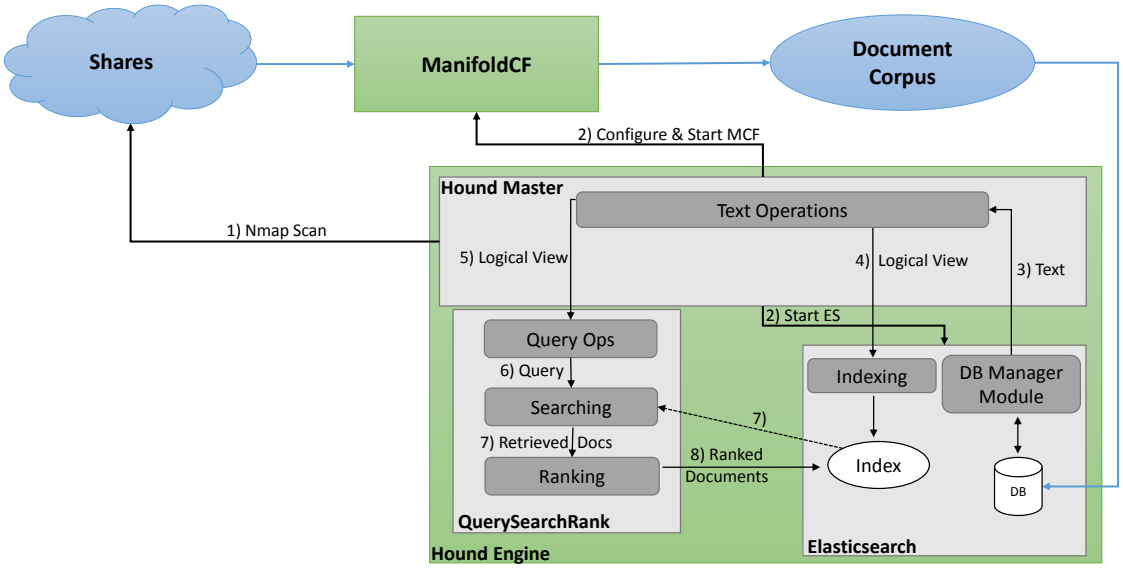

Figure 3.1: Hound System

As we can see in figure 3.1, Hound Engine corresponds to the IR process (figure 2.3), with slight changes made to better represent the Hound Search Engine. There is no User Interface, since the system is automatic. This Engine is composed by the Hound Master, which is the one responsible for scanning the network (step 1) and triggering the action of

* *A hound is a dog of a breed used for hunting, especially one able to track by scent. This project has the same function, which is to "sniff" for documents and report if it matches what it was searching.*

the auxiliary tools (ManifoldCF and Elasticsearch), configuring MCF automatically (step 2). We also have the Text Operations, that changes the raw text to prepare the search (steps 4 & 5). Besides the Hound Master, we have the following steps in QuerySearchRank: 1) Query Operations, that perform the query defined *a priori* in the system (step 6); 2) Searching, which is made over the text obtained from the Index (it is different in this case comparing to the original IR Process because here, the data from the index goes to the Hound Master and only then to the Searching module) (step 7); and 3) Ranking, that applies a classification algorithm over the retrieved documents, storing the ranked documents back into Elasticsearch in a new Index (step 8).

In the Elasticsearch section, we have 2 components: 1) the DB Manager Module (DBMM), that manages the connections to the database and the data that comes from it (step 3) and 2) Indexing, which stores the data into an Index of Elasticsearch.

3.3 Implementation of the Hound System

In order to retrieve the information from the repositories, it is necessary to know where to look. So the first step is for Hound Master to run a bash script that uses Nmap to scan the network and perceive which hosts contain open ports according to each type of network share. Then SMBclient is used with the purpose of finding out which of the IPs with open ports actually contain open shares with files.

The next step, which is the extraction of the files content, was made in two different ways during the project. The first uses a SMBclient to extract every file on each open share. The second uses ManifoldCF in order to automatically do it. This latest option is the best and it was the chosen way of retrieving the files content from each share, because it does everything by itself and has got many implemented connectors to extract the files from the repositories. For the use of this tool, the Hound Master configures ManifoldCF and triggers its action, alongside with Elasticsearch. ManifoldCF then stores the information of each file (metadata and content) in Elasticsearch for further searching (the content is stored with Base64 encoding).

The Hound Engine is what differentiates this solution from the others available in the market. Hound Master gets the raw text that was previously indexed and pre-process it for the searching. This pre-processing is composed of the following steps: 1) decode the text from base64, because the content is encoded; 2) iterate through each line of the content and then send it to the other module for Searching.

In the Searching step, the queries are applied (these queries are the type of information that we are looking for - phones, VAT, passwords, etc.) and the searching is made to check if each line contains the sensitive data that we want to find.

Having the information found, the Ranking of documents is then performed, according to the ratio of sensitive information that was found (or not).

Hound generates the following indexes:

- **hound_mcf** is a temporary index created by ManifoldCF, where the content of the extracted files is stored.
- **hound_terms** is where the sensitive words found in files is stored. The way this terms are classified and stored will be explained in detail in the next subsection.
- **hound_files** is where a summary of the files with sensitive information is stored. The way this files summary is made and stored will be explained in detail in the next subsection.

3.3.1 Hound Master

This component of the Hound Engine is the crawling mechanism responsible for scanning the network, triggering the auxiliary tools (ManifoldCF and Elasticsearch) to start running and pre-process the text for the Hound QueryScanRank module.

The network scan is performed by a bash script that uses Nmap and SMBclient connections. Nmap was used to find IPs with ports 139 and 445 open (Windows Share), with the command in listing 3.1 and the parameters in table 3.1.

Listing 3.1: nmap command

```
sudo nmap —open –n –T4 –Pn –p139,445 –sS –oA <output_path><ip_range>
```

Command	Description
--open	lists only the open port
-n	No DNS resolution. Tells Nmap to never do reverse DNS resolution on the active IP addresses it finds. This option slashes scanning times.
-T4	set timing template from 0 to 5 (higher is faster)
-Pn	no ping, treat all hosts as online. This option skips the Nmap discovery stage altogether. Scan hosts even if protected by firewall.
-p139,445	scan only specific ports (139 & 445)
-sS	TCP SYN scan. It can be performed quickly, scanning thousands of ports per second on a fast network not hampered by restrictive firewalls. It is also relatively unobtrusive and stealthy since it never completes TCP connections.
-oA	output scan - Nmap + XML + GNMAP
<output_path>	where the path to output results is specified.
<ip_range>	the range of ips to scan. Ex: 1.1.1.0/23 ...

Table 3.1: Nmap command parameters

Having the Nmap result, the following grep command was used to just keep the list of IPs:

Listing 3.2: grep command

```
grep –B3 open ${CURRENT}/<file>.nmap | grep report | awk '{print $5 }'
```

where *CURRENT* is the path of the folder where the script is running and *ip_* range is the range of IPs, same as in the Nmap command. With this list, I make smbclient connections are made to list the indexable folders existent in the IPs, using the following command:

Listing 3.3: smbclient command

```
smbclient -L //${IPS[${i}]} -A ${CURRENT}/authfile
-U <user> 2>&1 | egrep -v "ADMIN\\$|.\\$|print\\$|
--------|^$|Sharename       Type      Comment" |
egrep "Disk" | awk '{print $1}'
```

At this point, there is a clean list of IPs that I want to connect with ManifoldCF, therefore it needs to be configured. For that, I use ManifoldCF's Control by Servlet API (`https://goo.gl/PTazkq`), using every command needed for each connection.

The configuration of ManifoldCF is done automatically with the list if IPs obtained in the previous scan. Then, the jobs are started in order to extract the contents of the files in the database, indexing the text in Elasticsearch through the Output Connection.

Afterward, Hound Master gets the text from the Index (represented in figure 3.2), in order to preprocess it.

Figure 3.2: Document indexed by ManifoldCF in Elasticsearch

The preprocessing of the information is composed of three steps:

1) the decoding from Base64 (listing 3.4):

Listing 3.4: Base64 decoding

```
@content = Base64.decode64(info['_content'])
```

2) encoding some files that are not in UTF-8 format, forcing the encoding to ISO-8859-1 and then to UTF-8 (listing 3.5):

Listing 3.5: UTF-8 encoding

```
if !line.valid_encoding? || line.encoding.name.eql?("ASCII-8BIT")
  line = line.force_encoding("ISO-8859-1").encode("UTF-8")
end
```

3) splitting the content into lines for further searching, iterating for each line of the content.

Once the preprocessing is complete, the lines are used in the next module - Hound *QuerySearchRank* - where the queries are made to find the sensitive information.

3.3.2 Hound QuerySearchRank - Query and Search

The Querying, Searching and Ranking of the information are the most important and complex steps of this project, because it is a challenge to find information by patterns. In this subsection the Query and Searching processes are described and the next subsection will detail the Ranking.

The Queries are made in the Hound *QuerySearchRank* module over each line at a time, using the plugins to Search for sensitive information, i.e. potential passwords, phone numbers, VAT numbers, dates, marketing plans, name, email or IP, through regular expressions. An example of a plugin is shown in appendix (listing A.2). All the potentially sensitive data is returned to the Hound *Master*, which in turn stores the information in index *hound_terms* (listing 3.6).

Listing 3.6: *hound_terms* document example

```
"_source": {
  "term": {
    "value": "96XXXXXXX",
    "info": [ {
      "type": "phone",
      "subtype": "meo"
    } ]
  },
  "file": {
    "url": "file://///XX.XXX.XX.XXX/____/_________.docx",
    "filename": "_________.docx",
    "ip": "XX.XXX.XX.XXX",
    "lines": [ "102" => \t_______96XXXXXXX_____\n"],
    "digest": "56d5ff58dd075b167c14cb0b1656ceb4ee3ccbb6",
    "index_date": "2015/04/20 14:40:58"
  }
}
```

All the Query Plugins have two options of checking the data (table 3.2). The default mode uses regular expressions or algorithms and the other way, uses lists of predetermined words that the user specifically wants to find (this lists are stored in a database folder). The Name and Marketing plugin are the exceptions. The first because it uses a regular expression with a list of all the Portuguese names but also works with a specific list of

names. The second because it only use lists of words. In the following subchapters, all the plugins are described in detail.

	Regular Expressions	List of words
Phone	X	X
NIF/VAT	X + Algorithm	X
Password	X	X
Marketing		X
Name	X + List	X
Date	X	X
Email	X	X
IP	X	X

Table 3.2: Hound Plugins Operation

3.3.2.1 Phone Query Plugin

In this plugin, there are 2 main functions: *validate* and *getSubtype*. The first receives the string from *Hound QuerySearchRank - Query and Search* as input and applies a Regular Expression (listing 3.7) to check if it contains phone numbers.

Listing 3.7: Phone Regular Expression

```
/(?:  |\t|[a-zA-Z]|\W\D|^)(?:\+?0?0?351)?
((?:(?:2[1-9][0-9]{7})|(?:9[1-4,6][0-9]{7})))
(?:\\|  |\t|[a-zA-Z]|\W\D|$)/
```

The regular expression can be divided in three parts, as shown in the figure: the first is the content before the number, which can be a space, a tab, a letter a symbol or the beginning of the string, followed by a possible indicative +351, 00351 or 351. Then the second part is the number itself, that can be a telephone (beginning with 2, followed by a number between 1 and 9, and then any 7 numbers) or mobile phone (beginning with a 9, followed by a number between 1 and 4 or 6 and then any seven numbers from 0 to 9. The last part is what shows after the number, which can be a backslash, a whitespace, a tab, a letter, a symbol or the end of the string. The second function is used in *Hound QuerySearchRank - Query and Search* to get the types of phone, which are the different Portuguese mobile operators (meo, vodafone, nos, and phone-ix) or landline telephones.

3.3.2.2 NIF Query Plugin

This plugin also has the same two main methods, *validate* and *getSubtype*. In the first, it gets a string and if the validator finds any VAT values through the regular expression (listing 3.8) and the algorithm (listing 3.9), then returns an array with all the VAT values found.

Listing 3.8: NIF Regular Expression

```
/(?:\s|\t|[a-zA-Z]|\W\D|^)([125][0-9]{8})(?=\\|\t|[a-zA-Z]|\s|\W\D|$)/
```

Listing 3.9: NIF Algorithm

```
NIF has got 9 digits, being the last the control digit.
To calculate the control digit:
  – Multiply the 8th digit by 2, the 7th digit by 3,
    the 6th digit by 4, the 5th digit by 5,
    the 4th digit by 6, the 3rd digit by 7,
    the 2nd digit by 8 and the 1st digit by 9
  – Add the results
  – Calculate the Modulus 11 of the result, i.e., the
    rest of the division of the number by 11.
  – If the rest is 0 or 1, the control digit will be 0.
    If it is another number x, the control digit will
    be the result of 11 minus x.
```

The second method is used in *Hound QuerySearchRank - Query and Search* to get the type of VAT found (singular person, collective person, etc.), using a list of prefixes.

3.3.2.3 Password Query Plugin

"Passwords are like underwear; you don't let people see it, you should change it very often, and you shouldn't share it with strangers" – Chris Pirillo

This plugin was without a doubt the most challenging of all, and the reason is obvious - How to find a password in the middle of a text? Even now, I believe it is not possible to find 100% of the passwords in a file, however I tried to get as close as possible. Taking into account that lots of false positives would come up, I have made six regular expressions, one for each type of password, to have as less false negatives as possible:

1) passwords that appear with usernames from PT's domain, which looks like this: domain\\username:Pa55word.123 (listing 3.10);

Listing 3.10: PTP normal users password

```
/(?:^|  |\\t|\t|)((?:(?:#{domain})(?:\\{1,4}))(?:[a-zA-Z0-9]{5,15}))
        (?:(?:\s*(?:\=|\:)\s*)|(?:\\t)|(?:\\=)|(?:\\\\=))
((?=[\S]*[a-z])(?=[\S]*[A-Z])(?=[\S]*[^\w\d\s])(?=[\S]*[0-9])[\S]*)
        (?:  |\\t|\t|\\n|\n|$)/
```

2) passwords appearing in excel files as lists of passwords, separated by tabs (listing 3.11 and listing 3.12);

Listing 3.11: Excel Passwords

```
/^|(?:\t|\\t|\\\\t?)([a-zA-Z0-9]{5,20})(?:\s?(?:\\t|\\\\t|\t|,|:)\s?)
        ([a-zA-Z'"?!.$#%0-9]{5,16})(?:\\n|\\\\n)?$/
```

Listing 3.12: Excel Passwords 2

```
/^([a-zA-Z0-9]{4,20})\s([a-zA-Z0-9?!.$#%@]{4,16})(?:\s[a-zA-Z0-9]+)?$/
```

3) General passwords, with the format username:password (listing 3.13). The symbol that separates the username from the password can be anything, so I defined as possible the ':', '=', '\\=' or '->' with 0, 1 or more whitespaces in between.

Listing 3.13: General Passwords

```
/(?:\\t|\s|\t|^|[\W])([a-zA-Z0-9]{4,20})
(?:(?:\\=)|(?:\\\\=)|(?:\s?[\=\:]\s?)|(?:->))
([a-zA-Z0-9\\\'\"\?\!\.\$\#\%\@]{4,20})(?:\s|\\t|\t|\\n|\n|$)/
```

4) UserPass passwords, with the format 'username: "user" password "pass"', or just pass(word) "pass" (listing 3.14).

Listing 3.14: UserPass Passwords

```
/(?:(?:u|U)ser(?:name)?[^\w\d]+(\S+)\W)?
(?:p|P)ass(?:word)?[^\w\d]+(\S+)/
```

5) TwoColPass passwords, when showing in files with only two columns, with the format username password (listing 3.15).

Listing 3.15: Two Collumn Passwords

```
/^([a-zA-Z0-9]{4,20})\s
([a-zA-Z0-9\\\'\"\?\!\.\$\#\%\@\-\=]{4,20})(?:$|\n|\\n)/
```

6) Lastly, Complex passwords, which contains at least one lowercase character, one uppercase character, one digit and one symbol (listing 3.16).

Listing 3.16: Complex Passwords

```
/(?!.+[\/].+)(?:^|  |\\t|\t)
((?=[\S]*[a-z])(?=[\S]*[A-Z])(?=[\S]*[^\d\w\s])(?=[\S]*[0-9])[\S]{4,15})
(?:  |\\t|\t|\\n|\n|$)/
```

These regular expressions may not all be used, because first it checks if there are any PT user password. Then, if there is none, it passes through the excel regexps, should the file be of .xls(x) or ods extension. If it is not of excel extension, it goes through the other password regexps. Lastly, the string passes by the complex password regexp. The results are returned in a hash to be used in *Hound QuerySearchRank - Query and Search*.

3.3.2.4 Name Query Plugin

Although this is a simple plugin, it is the slowest to run. The reason is that it loads a text file containing the list of all the Portuguese names (2734 in this list, except for surnames) [15], which is a considerable amount of data. However, near the end of the project, we came with a new way of searching for names, which is to use a regular expression (listing 3.17) that have a parameter 'names', which is a string with all of the names of the file, separated by a pipe (|), meaning OR in regular expressions. The main function is *validate* and it gets a string, matching the regular expression to find the names. If any names are found, they are stored in an array and returned to be used in *Hound QuerySearchRank - Query and Search*. The function *getSubtype* is similar to the ones in the other plugins.

Listing 3.17: Name Regular Expression

```
/\W+(#{names})(?=\W)/i
```

3.3.2.5 Email Query Plugin

For the Email plugin, the process is similar to the other validators, with a regular expression (listing 3.18) that matches each line, storing the results found (there can be more than one in each line) in an array and then return it to be used in *Hound QuerySearchRank - Query and Search*. An example of a match with this expression is e.g. john@snow.com.

Listing 3.18: Email Regular Expression

```
/((?:[a–zA–Z0–9\.\–\_\+]+)(?:\@)(?:[a–zA–Z\–\_]+\.)(?:[a–zA–Z]{2,3}))/
```

An email is composed of 3 parts: the first, before the @ that can have letters, numbers, dots, hyphen, underscore, and plus; the second which is the @ symbol; and the last which is the domain of the email, that contains letters, hyphens or underscore, followed with a dot and lastly the top-level domains that are 2 or 3 letters.

3.3.2.6 IP Query Plugin

In the IP plugin, the process is the same as the one described above, only with a different regular expression (listing 3.19) to match each line. The regular expression is composed of 4 groups of numbers that can have from 1 to 3 digits and are all separated by dots. The format is e.g. 127.0.0.1.

Listing 3.19: IP Regular Expression

```
/(?:\s|[\D]|\=|\:)([0–9]{1,3}\.[0–9]{1,3}\.[0–9]{1,3}\.[0–9]{1,3})(?:\s|\\|\D|\t|\n)/
```

3.3.2.7 Date Query Plugin

Finding dates is not easy and the reason is because there is no standard date format for all countries. Some files have DD/MM/YYYY but others have MM/DD/YYYY, which makes the parsing harder because there is no way of knowing for sure if one or another are correct. Figure 3.3 shows the formats used in different countries and as we can see, there is a wide variety of date formats [13].

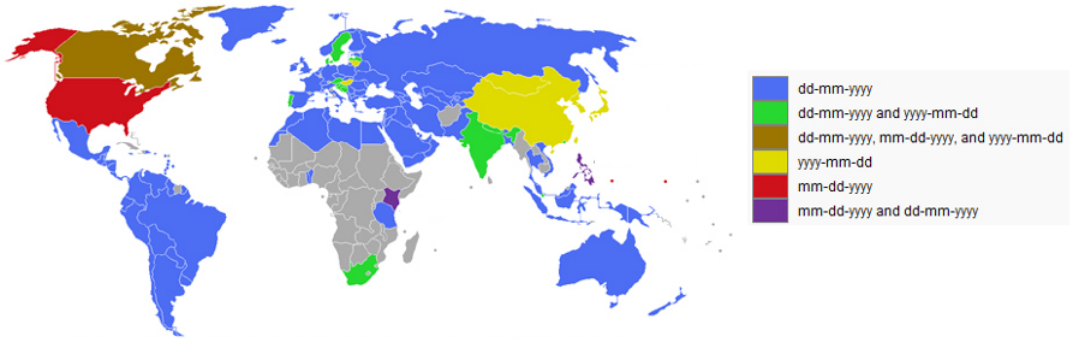

Figure 3.3: Date format by country

In order to find a solution for this problem, W3C defined a standard - The ISO date format. The international format, defined by ISO (ISO 8601) [16], tries to address all these problems by having a numerical date that looks like YYYY-MM-DD, where:

- YYYY is the year [all the digits, i.e. 2012]
- MM is the month [01 (January) to 12 (December)]
- DD is the day [01 to 31]

For example, "3rd of April 2002", in this international format is written as: 2002-04-03 [31]. Although this standard is out since 1998, the truth is that there are still too many date formats in the world. Embracing this format would, in my opinion, solve the problem. The validation of dates is made with a regular expression (listing 3.20) that receives a string and returns an array with the matches if they exist.

Listing 3.20: Date Regular Expression

```
/([0-9]{1,2}[.:\/-][0-9]{1,2}[.:\/-][2]{1}[0-9]{3})|
([2]{1}[0-9]{3}[.:\/-][0-9]{1,2}[.:\/-][0-9]{1,2})/
```

3.3.2.8 Marketing Query Plugin

The Marketing plugin is simple, but there is no easy way of defining it, at least in long-term. The problem here is that Marketing plans contain information of new products, which are its name, features and release date. Regarding the name and features, there is nothing we can do a priori, so the only way possible, in my opinion, is through a list of words stored in a file that needs to be changed over time, with specific information that we want to find. As for the release date, we can do something about it, which is to search for a date in the future. For that, we use the *Date Query Plugin* to verify if a line contains a date. If so, then it uses the function *compareDates* to compare the date and today's date.

3.3.3 Hound QuerySearchRank - Ranking

At the same time that the terms' information is stored in a hash, there is another hash created that contains information of each file and is updated when the terms are found. During the project, this last hash used to be inserted in Elasticsearch at the end of the program execution, in index hound_files, but later on, it came to be indexed at the end of each file's analysis. The reason for this change is that previously, if there was a failure on the program execution, the information of all the processed files would be lost. Now, the only thing that can be lost is the result of one file's processing. This improvement, allows not only to save time, but also to ensure that it does not lose information. The index is basically a resume of all the information found in each file. It is composed with the occurrences of each data type, the number of words and lines of the file, as well as the final classification of the file according to its sensitivity degree and its corresponding level, ending with a digest (made with SHA-1) and the date of the indexation.

An example of a document in hound_files index is shown in figure 3.21.

Listing 3.21: hound_files document example


```
"_source": {
  "ip": "XX.XXX.XX.XXX",
  "filename": "_________.docx",
  "url": "file:////XX.XXX.XX.XXX/____/_________.docx",
  "phone": [ {
          "subtype": "meo",
          "list": [ "96XXXXXXX" ],
          "count": 1,
          "percentage": 0.6451612903225806
  } ],
  "nif": { },
  "date": { },
  "name": { },
  "password": { },
  "email": { },
  "ip": { },
  "words_count": 5899,
  "lines_count": 155,
  "ranking": {
    "classification": "High Sensitive",
    "level": "Green"
  },
  "digest": "56d5ff58dd075b167c14cb0b1656ceb4ee3ccbb6",
  "index_date": "2015/04/20 16:21:22"
}
```


The Ranking of the file, as already mentioned before, is made by two metrics: 1) the Classification according to its sensitivity (High Sensitive or Sensitive) and 2) the Level of the classification (Green, Yellow, Orange, or Red), according to an average of two parameters: 1) the ratio of the occurrences of the sensitive data and the total number of lines (not counting the white lines) of the file and 2) the number of lines of the files. With this ratio, a number from 1 to 4 is assigned according to the ratio obtained (1 - 0..19,9%; 2 - 20..39,9%; 3 - 40..69,9%; and 4 - 70..100%). As for the number of lines, a value from 1 to 4 is also assigned if the quantity of lines is less than 60, from 60 to 200, from 200 to 400 or more than 400, respectively. Then, the average is calculated with these two values, giving the final number, which corresponds to the color referred above. The first value has a higher weight than the second, in case the average is a half value like 1,5.

In the example above, we have a file that contains one occurrence of a cellphone of the brand MEO. The information was manipulated to show less data because of the information's sensitivity and in order to fit, otherwise it would be too extent. The classification of the file is High Sensitive because it contains phone numbers and the level of severity is Green because the percentage of phone numbers in this file is lower than 1%, so it gets the value 1 and as the number of lines is 155, the second value is 2. Given that the average gave 1.5 and since the first value has a higher weight than the second, the final average value is 1, which means Green.

3.4 Hound Integration to PT's platform

There is a large number of technologies used by PT's DCY in order to support their platform. Since the objective of this project was to create a solution that would become a component to be integrated in this platform, it is important to identify the existing technologies in it. The relevant ones for this project, are the already mentioned Elasticsearch and Kibana, as well as RabbitMQ. Many other technologies are used for data analysis and statistic generation like AllegroGraph and OpenTSDB, however they are not relevant for the integration of this solution and for that reason, they will not be described.

RabbitMQ is a messaging broker (fig. 3.4) - an intermediary for messaging, giving applications a common platform to send and receive messages, which enables software applications to connect to each other, as components of a larger application. Its messaging is asynchronous, decoupling applications by separating sending and receiving data.

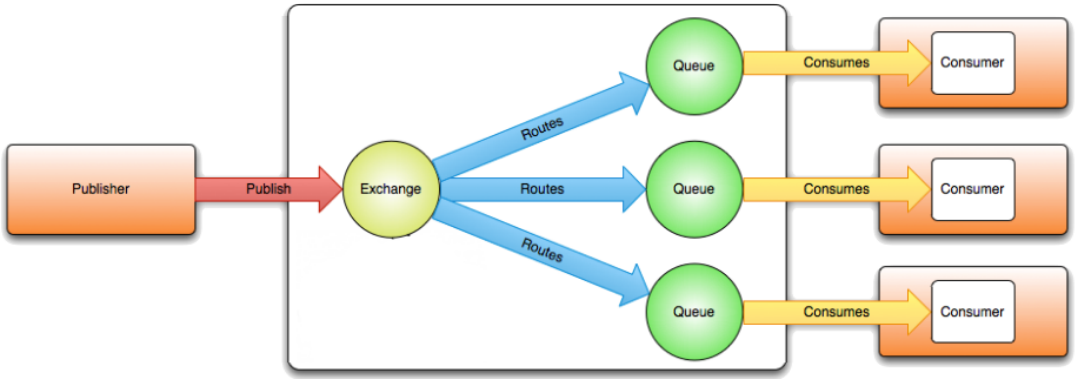

Figure 3.4: RabbitMQ queue example

Having Hound completely implemented, it was necessary to adapt it in order to integrate with PT's platform (recently called Hidra - High Performance Infrastructure for Data Research and Analysis). Figure 3.5 shows the final architecture of Hound, integrated with Hidra infrastructure, having 3 instances running in different machines.

Hidra receives data through an AMQP Broker (RabbitMQ), in a format defined in a custom Ruby Gem designed by PT, pushing the data to a queue. The consumption of this data is made through a Publish/Subscribe scheme, where subscribe gathers the data from the queue and publishes in the correct format to the desired database (in Hound's case, it is Elasticsearch, but there's an OpenTSDB database in Hidra for other projects). The front-end of the Elasticsearch database is Kibana, which provides customized graphics for each use-case, in order to have a better visualization of the stored data.

Hidra contains many components, and Hound is expected to be another one, with its full integration to the platform. The mission of Hound is to search PT's network and find sensitive data, which is sent to the Elasticsearch used in Hidra. This data will further be

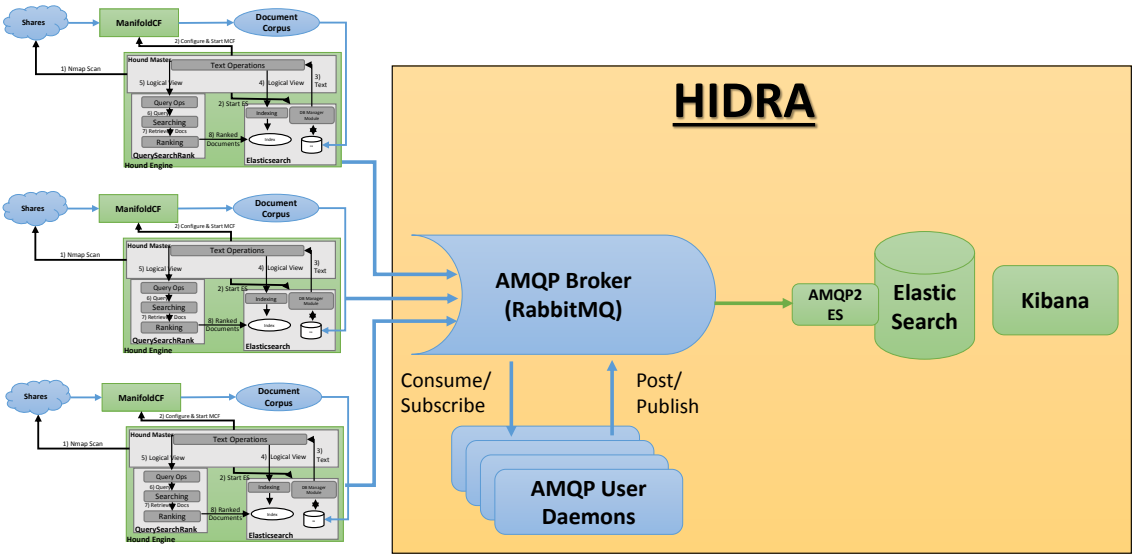

Figure 3.5: Hound integrated in Hidra platform

analyzed and soon, there should be implemented a machine learning algorithm to increasingly have more refined data. There can be as many instances running as needed, because the data is sent to Hidra's queue and there will be no problem regarding information concurrency, thanks to the RabbitMQ functionalities.

As we can see, some changes were made regarding the data storage. Instead of storing everything to the local Elasticsearch as before, with this integration, only the information gathered by ManifoldCF is stored there, whereas the processed information is directly sent, through the ruby gem, to the Elasticsearch in Hidra. The local ES has now only one index (hound_mcf), working just as a temporary instance, while the ES in Hidra have two indices (hound_terms and hound_files). The designation of the indices is built with my environment in PT and the associated domain, along with the original name of the indices, respectively for each index. For hound_files, it would be *environment.domain_hound_files*.

3.5 Chapter conclusion

This chapter presents all the work carried during the internship in PT Comunicações, with a detailed presentation of the tools and the way in which they have been used (Nmap, Smbclient, Elasticsearch and Apache ManifoldCF), as well as an in-depth description of the engine created in Ruby. As we have seen, Hound Engine is composed of Hound Master and QuerySearchRank. Hound Master is responsible for the scan in the network, start the execution of ManifoldCF (and configure its connections) and Elasticsearch, as well as preprocessing the indexed data. The queries, searching and ranking are assigned to Hound QuerySearchRank, where it receives the preprocessed information from the Master

and searches for the presence of sensitive terms with the help of the developed plugins, one for each query (the type of information that we are searching for). It also ranks the analyzed files, creating "reports" in JSON format, that are stored in a new Elasticsearch index. We achieved the purpose of creating an incremental and scalable system (as shown in appendix lst. A.1), being easy to add new kinds of data that we intend to find.

After the final implementation, Hound was then integrated into Hidra (the platform of DCY), being modified in order to operate as one of its component. The modifications were few, so there were no serious issues. In the next chapter, there will be an evaluation made to the system, in order to understand its overall performance and the quality of the data gathered.

Chapter 4

Evaluation

4.1 Prototype

For the prototype, it was used an ASUS Laptop with a Core i7 2.40GHz processor and 8 GB of RAM, in a Xubuntu 14.02.2 LTS distribution (Ubuntu + XFCE).

Nmap (v6.40) and ManifoldCF (v2.1) were used, and the last required some changes to work as needed. For that, it was necessary to make some modification on the connectors file (connectors.xml) to use the windows share connector (disabled by default), download the jcifs plugin (jcifs-1.3.18.jar) that was placed at "connector-lib-proprietary" folder; and change the configuration file (figure 4.1) to use PostgreSQL instead of HSQLDB.

```
<property name="org.apache.manifoldcf.databaseimplementationclass" value="org.apache.manifoldcf.core.database.DBInterfacePostgreSQL"/>
<property name="org.apache.manifoldcf.database.name" value="*********"/>
<property name="org.apache.manifoldcf.dbsuperusername" value="********"/>
<property name="org.apache.manifoldcf.dbsuperuserpassword" value="***********"/>
<property name="org.apache.manifoldcf.hsqldbdatabasepath" value="."/>
<property name="org.apache.manifoldcf.database.maxhandles" value="100"/>
<property name="org.apache.manifoldcf.crawler.threads" value="50"/>
<property name="org.apache.manifoldcf.crawler.historycleanupinterval" value="2592000000"/>
```

Figure 4.1: ManifoldCF properties.xml

After that, in order to obtain a better performance, I made some changes to the configuration file of Postgres Database (table 4.1).

shared_buffers	1024 MB
checkpoint_segments	300
maintenanceworkmem	2 MB
tcpip_socket	true
max_connections	200
checkpoint_timeout	900
datestyle	ISO, European
autovacuum	off

Table 4.1: Postgres configuration file - postgresql.conf

The used connectors were Elasticsearch Output Connector, Active Directory Authority Connector, Tika Transformation Connector, Windows Share and Sharepoint Reposi-

tory Connectors (where the list of IPs was used).

Elasticsearch (v1.6.0) was also modified so that it uses more RAM memory and reduce the time taken for each action.

Hound Engine was designed with Ruby (v2.2.2) and we used some Ruby gems such as chronic, curb, elasticsearch, json, mimemagic, etc.

4.2 Changes during the Implementation

Early in the project, I used some Ruby gems to parse the content when it was not immediately readable, like Yomu that uses Apache Tika content analysis toolkit for the files in Office, RTF, PDF, OpenOffice or other format and Nokogiri for the HTML and XML files. However I later discovered that it was possible to make this parsing in Manifold as it extracted the files content (using Apache Tika, which is the same as what Yomu uses), so I came to use this connector in ManifoldCF instead of the gems. To check the MIME type of the files, I resorted to a ruby gem *MimeMagic*, because it allows to know the file type not only by the extension of the file, but also by the content, which it is a very useful tool. However, there are some problems with this tool regarding some formats. For example, some office files are considered as application/zip mime and so it is necessary to classify the mime by extension.

Listing 4.1: MimeMagic gem

```
extension = fileName.split('.')[-1]
mime = MimeMagic.by_magic(@content)
if ((mime.to_s.size==0) || (mime.to_s.eql? "application/zip"))
   mime = MimeMagic.by_extension(extension)
end
```

Even though I ended up using the Tika Extractor Connector of ManifoldCF, I believe it is important to refer how the parsing is made with the ruby gems, because it works well and it was used during more than half the time of this project. The used gems were *Yomu* and *Nokogiri* as was referred before.

Listing 4.2: Yomu and Nokogiri gems

```
@content = Yomu.read :text, @content

data = Nokogiri::HTML(@content)
data = Nokogiri::XML(@content)
@content = data.content
```

With *Yomu*, I was able to parse the content of Office formats, OpenOffice OpenDocuments, RTFs and PDFs. This was helpful, because the content of these files is encoded, thus unreadable when you look at the content, instead of opening the file manually. *Yomu* allows reading the metadata and the content, but for the project's purpose, only the content was necessary. With *Nokogiri*, I was able to parse the content of XML and HTML pages a lot easier than if I had to build a parser by myself and it is a really great tool.

4.3 Evaluation Results

Evaluating a retrieval system is assessing how well it meets the information needs to its users. This section intends to evaluate the performance and quality of the data obtained by the Hound System in a real environment.

4.3.1 Performance of Hound

The evaluation of the performance lies on understanding if it has a good time of execution, if the resources used are not too many, and if the runtime grows proportionally to the increase of data. The performance of the main components/steps of the Hound System execution will be described in detail.

4.3.1.1 Bash Script

Using the Nmap tool to scan ports 139 and 445 in 512 IPs with SMBclient connection, took an average time of 37.5 seconds (9.5 seconds for Nmap and 28 for SMBclient) and found 334 hosts up. This time is an average of successive scans one after the other, where the time taken by the first execution was superior to the following because, in the first, the TCP connection needs to make the 3-Way-Handshake, so it takes a little longer. For the next executions, since the connection is still up, there is no need for the handshake, making it faster. However, since the Nmap scans will not be executed one after the other, but in the beginning of each Hound execution, the Handshake will always be made, so the considered average time of the first scans is 52 seconds for a range of 512 IPs.

The extra time that the SMBclient connections take, is compensated in the next step as we decrease the number of ManifoldCF connections to the minimum really needed (17 instead of 334, in average). Considering the open ports found during 8 weeks of tests, its maximum number was 21 and its average was 17. Nmap scans are considerably fast and they take half the time (4,7 seconds) if we do not output the content to a file.

4.3.1.2 ManifoldCF

For ManifoldCF, for the three ranges, we get the following indicators:

	Range of IPs 1	**Range of IPs 2**	**Range of IPs 3**
ManifoldCF Max RAM usage	1,4 GiB	2,1 GiB	1,7 GiB
Elasticsearch Max RAM usage	1 GiB	1,8 GiB	1,3 GiB
Postgres Max RAM usage	0,7 GiB	1 GiB	0,8 GiB
Max network speed	10 MiB/s	12 MiB/s	10 MiB/s
Average number of files found	3060	19045	4352
Average Time taken (with ArticleExtractor)	26 min	125 min	37 min
Average Time taken (with DefaultExtractor)	12 min	63 min	20 min

Table 4.2: ManifoldCF Performance

The performance of this tool suffered greatly because of one of its components - Tika Extractor. With this transformation connector, it is necessary to specify the Boilerpipe. By choosing the KeepEverythingExtractor, although the time taken is the highest of all the extractors used, the amount of information kept is also the larger. Later on, ManifoldCF found a file that led the CPU to use the most of its capacity, leading to a crash in MCF's engine. Having this problem, and after finding where it happened, since there was nothing that could be done to correct the problem, I have chosen the DefaultExtractor, which solved it. Besides, the time taken by ManifoldCF decreased by half with this boilerpipe, so it was a positive change. Even if the amount of data was higher with the previous boilerplate extractor, it was not relevant for the purpose of this project, so the DefaultExtractor became the best choice after all. The average times of executing this step differ according to the Extractor we use and the size of files that we are extracting. Even though ArticleExtractor takes more time, it achieves higher accuracy than DefaultExtractor. However, it also led to a crash, so I kept using the DefaultExtractor (figure 4.2). As for the files obtained, ManifoldCF retrieved the exact types that were specified, which were the .pdf, .docx, .xlsx, .pptx, .txt, .rtf, .log and a few other documents.

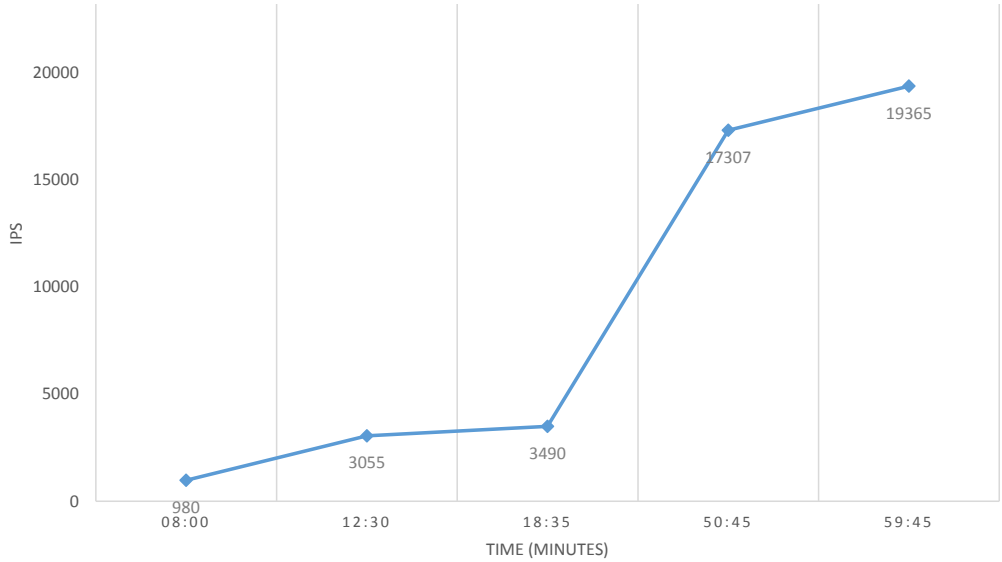

Figure 4.2: ManifoldCF time taken

4.3.1.3 Hound Engine

The Hound Engine used an average of 37,5 MiB in the beginning, which grew until the end of its execution. This happens because, during the execution of Hound, it stores information in cache that increases as each file is processed. This information are two hashes. One with the path and the SHA1 of the files and another that contains temporary information concerning the original files that are being processed. The execution time of the Hound Engine variates depending on the number and size of lines. Figure 4.3 shows the number of files processed by time, executed in the first range of IPs tested.

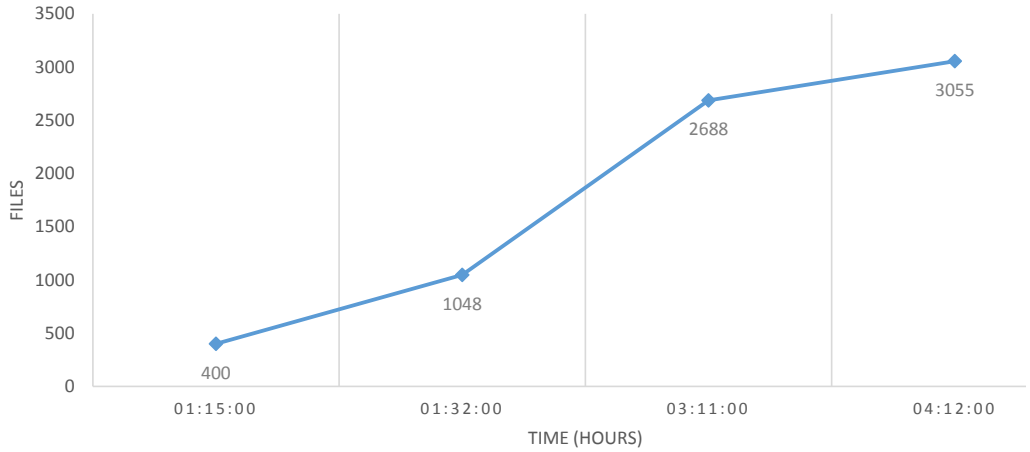

Figure 4.3: Hound Engine execution time - range 1

This first range contained an average of 3055 files, which took an average of 4 hours to process everything. Figure 4.4 shows the time taken to analyze and classify the files found on the second range of IPs.

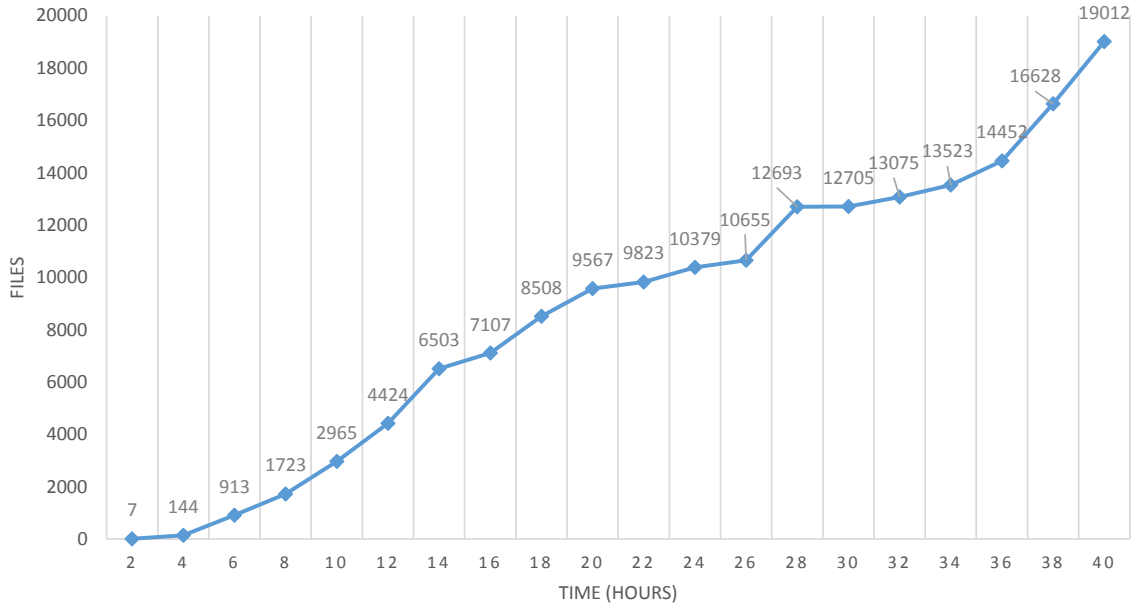

Figure 4.4: Hound Engine execution time - range 2

The second range contained many more files (an average of 19012), so the time taken was also higher than the previous. Moreover, the size of the files were bigger, some of which with around 900000 or more lines, which explains the increase of time taken. For example, between the 28 and 30 hours, only 12 files were processed, due to a file with 897452 lines, whereas between hour 36 and 38, it processed 2176 files. Figure 4.5 shows the time taken for the third range of IPs.

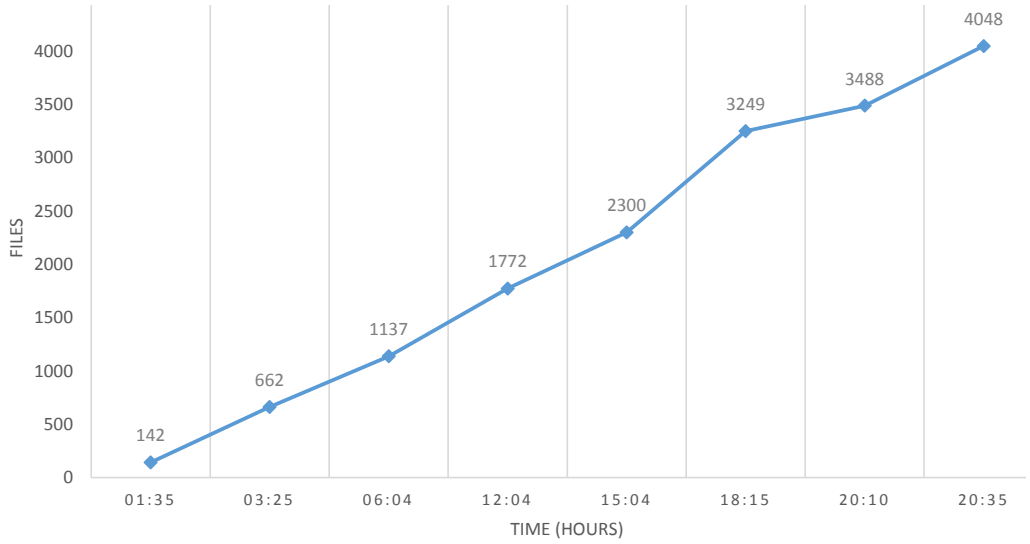

Figure 4.5: Hound Engine execution time - range 3

The third range contained practically the same amount of files as the first, although the size of the files were much bigger (some files had more than a million lines), so the time taken was also greater. Since there is no exact way to calculate the time required to process the information, figure 4.6 provides a more accurate way, by showing the variations of time taken, according to the number of lines in the files processed by second.

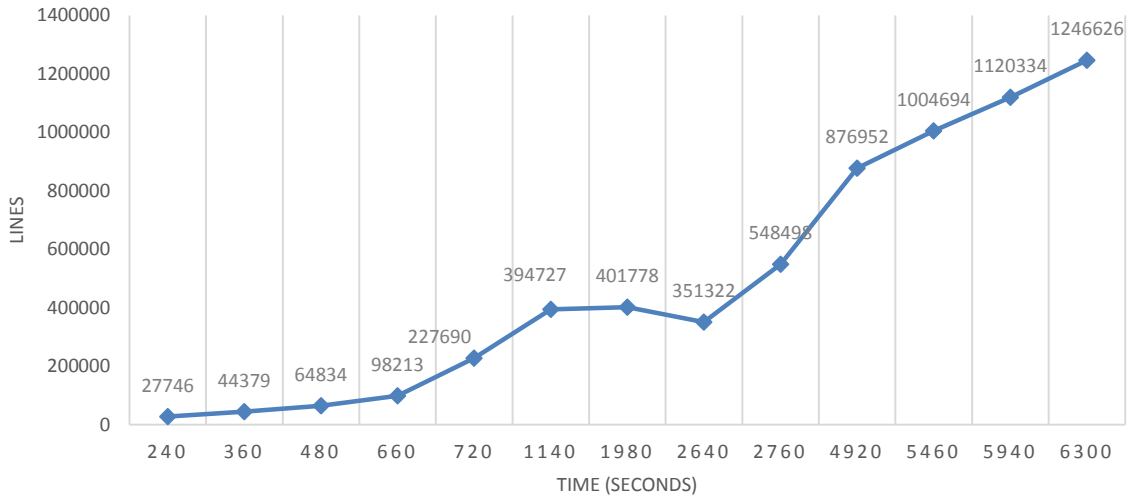

Figure 4.6: Lines processed by seconds

This chart shows the number of lines processed by second, which provides a more precise analysis than comparing file by file, as shown in the graphics above, allowing to make closer predictions of the time that a given file may take to be processed and classified. The data represented is a merge of the times gathered from the three IP ranges scanned. The maximum number of lines found were 1246626, which took approximately

1h45. With this information, I could reach a value that is not too far from reality, which is approximately 197 lines per second. Taking into account that for each file, every line passes through the regular expressions of each plugin (there are 8 plugins and a total of 12 regular expressions), 197 lines per second shows to be a good result.

The time taken and the RAM usage of Hound escalates linearly (in average), which is good because it shows that there are no scalability issues. This happens because after indexing the data, the local hash is deleted, so the only hash that increases is the one with the pair (file processed) $\Rightarrow$ (SHA1 of the file), which has no great impact to the overall performance of Hound. Also, an important factor is that Hound treats most of the exceptions that may occur during its executions, only breaking when it can loose the data that it has processed until that moment. Should a break happen during a file processing, it stores, in cache (temporary file), the information processed until that moment relative to that file, as well as the line where it stopped. When launching Hound again, it simply continues in the line where it stopped before, and keeps processing the content of the file as if there had been any stop, and more importantly, without any loss of information.

4.3.2 Quality of information obtained

Even though the components runtime is an important factor, the quality of the data found is even more, with the main objective to find a balance between false positives and false negatives (figure 4.7).

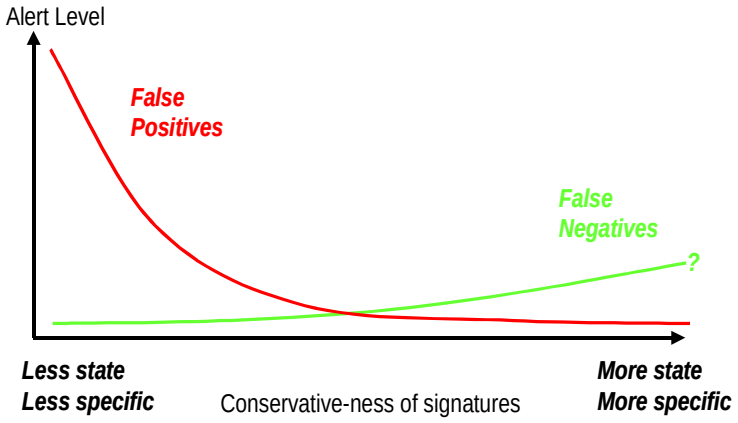

Figure 4.7: False positives vs false negatives, retrieved from [22]

This chart contains information related to IDS (Intrusion Detection Systems), where the relation between false positives and negatives is most important in order to realize which disturbances in the network actually correspond to intrusions. In the context of this project, false positives correspond to data classified as sensitive, but in fact are not. On the other hand, false negatives are the data that are not classified as sensitive, remaining

undetected. So the main concern lies in having the minimum percentage of false negatives possible, while having an acceptable number of false positives.

The number of occurrences of the types of information we intended to find, proved to be different in each scan. In figure 4.8 we can see the results of the first scan.

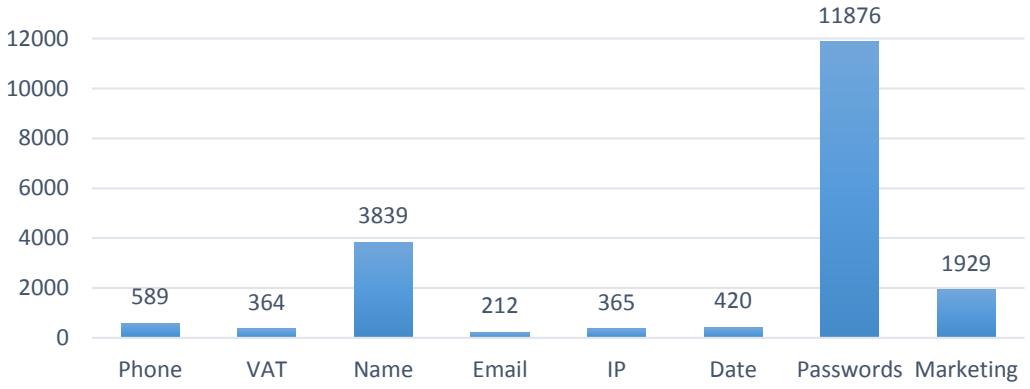

Figure 4.8: Occurrences of scan 1

The scanned files were around 3000, most of them being reports and software documentation. Some of them contained relevant information and the type of data that was mostly found was password related and the least was email. The password and marketing data were the least precise, having too many false positives. In figure 4.9 we have the results of the second scan.

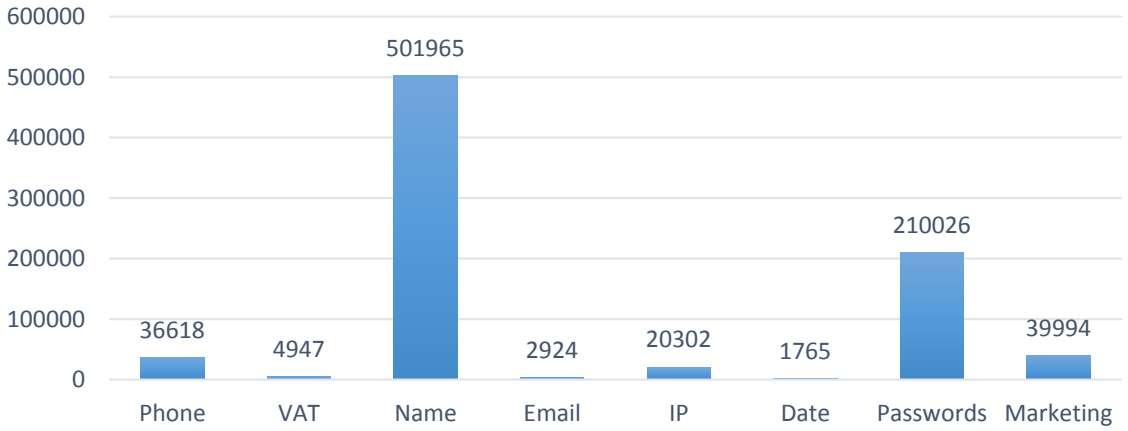

Figure 4.9: Occurrences of scan 2

This second scan was the richest with respect to the quantity and quality of the information collected. This scan found around 19000 files. Many names were found, where some of them (mostly names like job, gui, etc.) were false positives. Many passwords were found, where some from documentation were actual passwords, but no explicit list of passwords was found. The remaining information found was of great quantity and

quality, except for marketing terms, which was mostly composed with false positives. In figure 4.10 we have the results of the third scan.

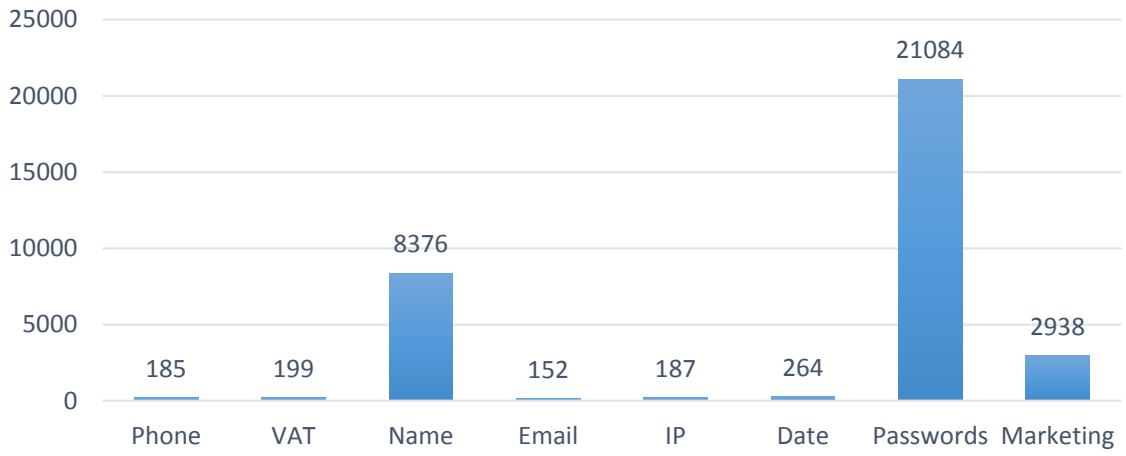

Figure 4.10: Occurrences of scan 3

In this third scan, we found around 4378 files, with a few more results than in the first scan. The occurrence of false positives also showed up similar to that observed in the first scan. The percentages of false positives obtained in Hound's executions, which were calculated taking into account the number of occurrences described above, are shown in figure 4.11.

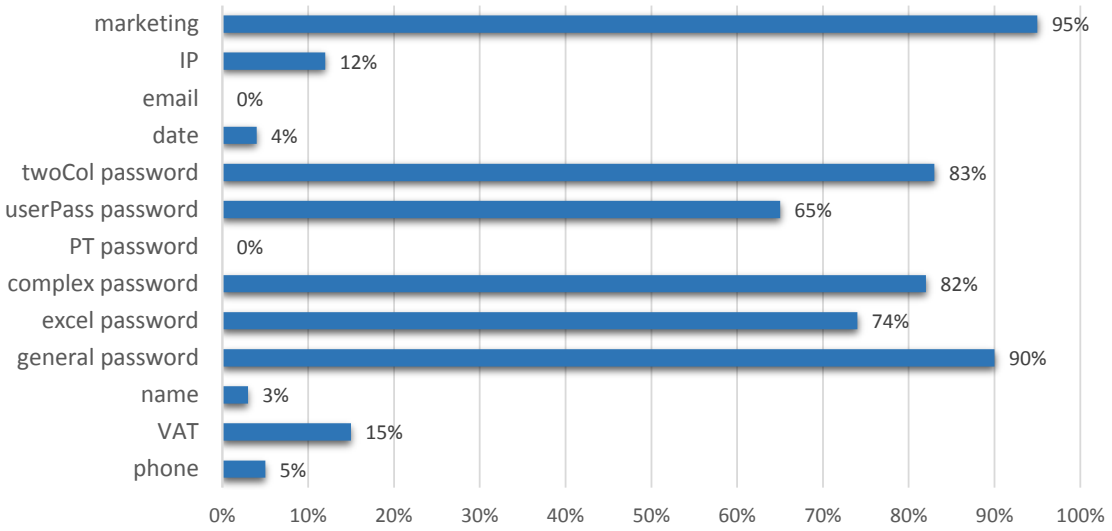

Figure 4.11: Percentage of False Positives

As we can see, we have obtained good results for nearly every query plugins, excluding the marketing and password plugins (except the case of PT passwords). The plugins that worked better were the email and the PT users password, due to its more specific

format, which eased the search for information. Phone, Date and Names got from 1% to 5% of false positives, which are good results. The IP plugin had around 12% of false positives, because when it found chapters or sections with 4 levels (e.g. 2.3.4.5) on documentation files, it considered it as an IP. As for VAT plugin, it had around 15% because, even though the number passed correctly through the algorithm of validation, the values were actually just a number with the same format as a VAT or a phone number (In Portugal, VAT and phone numbers have the same format, unless the phone is represented with the area code 351). However, the algorithm is well implemented so every VAT number existent in the files were found, which gave no false negatives.

The biggest problem was the marketing query plugin, where I could not find an effective way of finding marketing plans, unless it contains a future date. But even if this date is not relative to a marketing plan, instead it could just be a scheduled call to be made to a customer or the resolution of a technical problem of a costumer. As for general passwords, many false positives arose, because the format of the data matched with the regular expression is too common in files that have no specific format, especially in software documentation files. However, having these many false positives, allows to have a minimal value of false negatives.

After these tests to the three floors of PT, we made a scan to a test folder with files that mostly contained passwords, VATs, IPs, names and emails. The objective of this scan was to have a small and controlled test group, allowing an inventory of false positives and false negatives. The occurrences of terms found, and those which actually appear on the files are shown in figure 4.12.

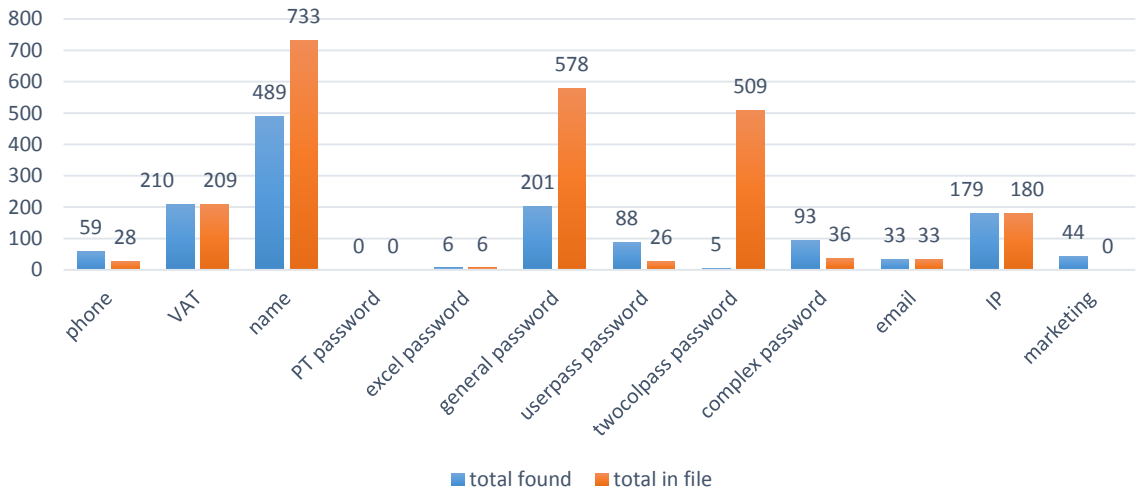

Figure 4.12: Terms found vs true terms in file

The percentage of false positives and negatives based on the occurrences represented above are shown in 4.13.

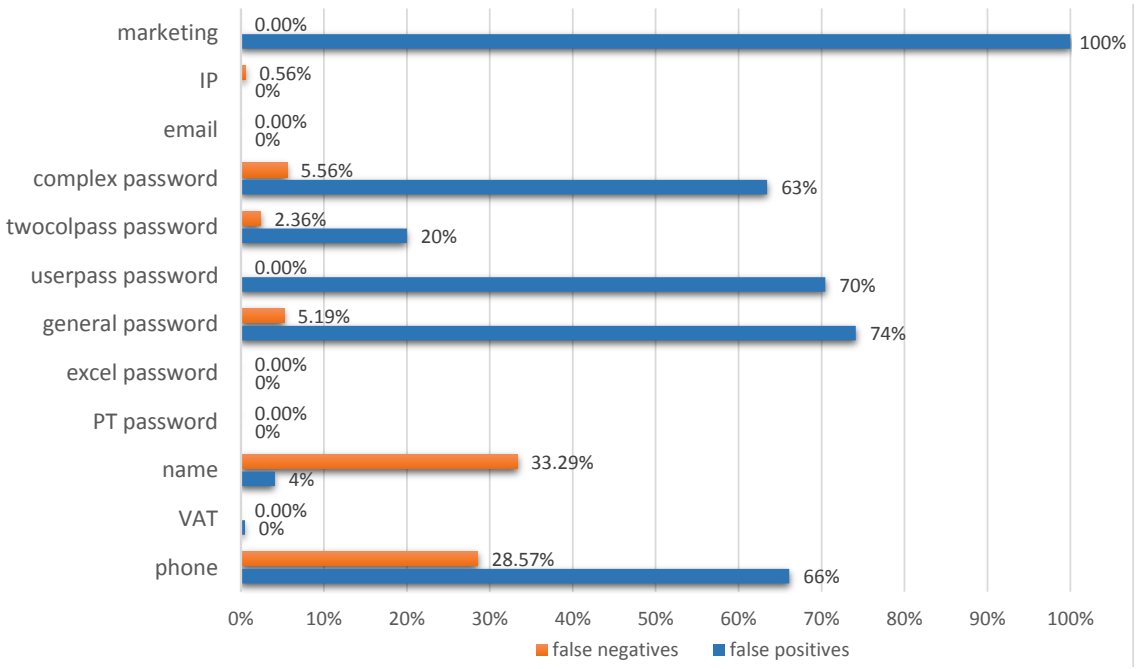

Figure 4.13: False Positives vs False Negatives

As we can see, we have found all the emails (33), all the excel passwords (6), and every VAT (210) with no false positives or negatives. The userPass passwords were all found (26), having 70% of false positives and the IPs were mostly found (179), with 4,76% of false negatives and no false positive. The generalPass (201), twocolpass (5) and complex (93) passwords were mostly found, with less than 6% of false negatives, however they had many false positives (74%, 20% and 63% respectively). The twoColPass got many false negatives due to a file that contained the 2014 most common passwords, where the first column corresponded to the count of the passwords, which were numbers with two or three digits (the size of the user that I am searching is at least 4, so it missed these data). More than half of the names were found (489), with 4% of false positives and 33% of false negatives. These false negatives represent surnames, which I do not have in my names list. The phones found (59) were correct for the normal format (914356789), finding all the phones in a list of 20. The exception were 8 phones (28,57% of false negatives) that were separated by spaces in groups of three digits (e.g. 211 345 567). These were not considered as phones, because the format was ignored on the regular expression and the reason is that, by catching this format of phone, it would highly increase the number of false positives. Besides, as there were many VATs found, some of them were also considered as phones, and that is the reason why 66% of false positives were found. The worst case proved to be the marketing data (44), which gave 100% of false positives, because there was no marketing plan in the test folder.

4.4 Chapter Conclusion

These scans provided a great quantity of information, which would be impossible if this thesis was not conducted in a so large company as PT is. Regarding the performance of the solution, we achieved a scalable system as it performs accordingly for any number of files (should they have a similar amount of lines). It does not require too many computer resources, except for when it is using ManifoldCF to extract the content of the files. As for the quality of the information, the system captures most of the data types proposed in the requirements, except for the marketing and passwords, where many false positives were found. However, there were not so many false negatives, which was the main objective of this system. In order to achieve this, we had to sacrifice the false positives, to grant that it should leave as less as possible of sensitive information uncaught.

The last test made with the local folder as a target, proved to be important to understand some flaws that the system had and still have. Some improvements were made during this test, and it was possible to make an analysis to the false positives and negatives, which would be impossible towards the more than 20000 files found in the three scans made to PT.

Chapter 5

Conclusion and Future Work

This report describes all that was done during the internship in PT, which was the research needed to provide the necessary background to carry out this project, its implementation thoroughly detailed, and the results with the performance and quality of the data gathered.

Regarding the research conducted, Information Retrieval was mostly followed by Baeza-Yates and Ribeiro-Neto's *Modern Information Retrieval* (1999) [1], not only because it was the most complete information that I have found about this subject, but also because almost every papers that I have read refer to this book. Regarding the Crawler, I have mainly relied on Batsakis et al.(2009) [2] for the starting understanding of a crawler's operation, and in Google's search engine to complete the learning and to observe the real case of one of the most used search engines in the world. Then, it was necessary to investigate and choose the required tools, like Elasticsearch and ManifoldCF. To pick the Elasticsearch, I have made some comparisons with Solr in order to understand which one would be the best solution for this project. As for Data Classification, the definition of Mike Chapple (2013) [6] was undoubtedly the most concise, simple and efficient, therefore it was chosen to be used and eventually adapted.

With respect to the work itself, an Information Retrieval system was designed, called Hound, where its Engine use Nmap to scan the netowrk, then it configures and use ManifoldCF to crawl the files in the internal network, indexing the information found in Elasticsearch. After that, it decodes the content from Base64, checks if the encoding is valid (utf-8) and iterates for each line of the files. These lines are then used in Hound QuerySearchRank module to search for sensitive information, using some implemented query plugins. After analyzing the files, a report is made with the ranking of each file, containing the sensitive data found and some other important information.

In order to improve Hound, it is possible to add as many plugins as needed, due to its extensibility. A possible way to refine the validation of data, would be to use some Supervised Machine Learning algorithms (since the data is already labeled after the content extraction and processing - which is basically a preprocessing).

Machine learning was not applied to this project for two reasons: the first is that

there was not enough time to fully understand it, due to its complexity; the second is that some Machine Learning algorithms will already be used in a future phase on the Hidra's platform. Data Mining and Machine Learning are fields that are emerging and growing really fast due to the increasing use of data verified these days, which imposes an urgent need of dealing with such massive amount of information available.

Even though it got some false positives in two query plugins, I believe Hound stands as a good solution and fulfilled its proposed objectives. It provides a new way of automatically finding and classifying files in an internal network and should provide DCY with the necessary mechanisms in finding sensitive and confidential information, which was not possible until now.

Bibliography

[1] BAEZA-YATES, R. A., AND RIBEIRO-NETO, B. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999. http://people.ischool.berkeley.edu/~hearst/irbook/1/node1.html.

[2] BATSAKIS, S., PETRAKIS, E. G. M., AND MILIOS, E. Improving the performance of focused web crawlers. *Data Knowl. Eng. 68*, 10 (Oct. 2009), 1001–1013. https://web.cs.dal.ca/~eem/cvWeb/pubs/Batsakis-Petrakis-Milios-DKE-2009.pdf.

[3] BELKIN, N. J., AND CROFT, W. B. Information filtering and information retrieval: Two sides of the same coin? *Commun. ACM 35*, 12 (Dec. 1992), 29–38. https://www.ischool.utexas.edu/~i385d/readings/Belkin_Information_92.pdf.

[4] BRIN, S., AND PAGE, L. The anatomy of a large-scale hypertextual web search engine. *Comput. Netw. ISDN Syst. 30*, 1-7 (Apr. 1998), 107–117.

[5] CERI, S., BOZZON, A., BRAMBILLA, M., VALLE, E. D., FRATERNALI, P., AND QUARTERONI, S. *Web Information Retrieval*. Springer-Verlag Berlin Heidelberg, 2013, ch. The Information Retrieval Process, pp. 13–26. http://www.springer.com/cda/content/document/cda_downloaddocument/9783642393136-c2.pdf?SGWID=0-0-45-1414724-p175333229.

[6] CHAPPLE, M. Data-classification levels for compliance: Why simple is best, Sep 2013. http://searchsecurity.techtarget.com/answer/Data-classification-levels-for-compliance-Why-simple-is-best.

[7] ELASTICSEARCH. Indexing a document. http://www.elastic.co/guide/en/elasticsearch/guide/master/index-doc.html. Accessed: 2015-04-21.

[8] ELASTICSEARCH. Open source distributed real time search and analytics | elasticsearch. Retrieved: April 21, 2015, from https://www.elastic.co/products/elasticsearch.

[9] ELASTICSEARCH. Scale horizontally. Retrieved: April 21, 2015, from http://www.elastic.co/guide/en/elasticsearch/guide/master/_scale_horizontally.html.

[10] GARFINKEL, S., AND SPAFFORD, G. Chapter 27.3: Can you trust people? In *Practical Unix & Internet Security, 2nd Edition*. O'Reilly Media, Inc., 1996. http://www.diablotin.com/librairie/networking/puis/ch27_03.htm.

[11] GARTNER. Now is the time for security at the application level, December 2005. Retrieved: November 17, 2014, from http://www.sigist.org.il/_Uploads/dbsAttachedFiles/GartnerNowIsTheTimeForSecurity.pdf.

[12] GOOGLE. Interest in solr and elasticsearch over time. Retrieved: November 20, 2014, from https://www.google.com/trends/explore#q=solr,elasticsearch.

[13] GUARDIAN, T. Why do americans write the month before the day?, 2013. Retrieved: April 24, 2015, from http://www.theguardian.com/news/datablog/2013/dec/16/why-do-americans-write-the-month-before-the-day.

[14] INKPEN, D. Information retrieval on the internet, 2014. http://www.site.uottawa.ca/~diana/csi4107/IR_draft.pdf.

[15] IRN. Lista de nomes admitidos. Retrieved: June 23, 2015, from http://www.irn.mj.pt/sections/irn/a_registral/registos-centrais/docs-da-nacionalidade/vocabulos-admitidos-e/downloadFile/file/Lista_de_nomes18-06-2015.pdf?nocache=1434623650.94.

[16] ISO. Iso 8601, 1988. Retrieved: April 24, 2015, from http://metric1.org/8601.pdf.

[17] LUCENE, A. Apache lucene - scoring, 2013. Retrieved: May 02, 2015, from https://lucene.apache.org/core/3_2_0/scoring.html.

[18] MIRANDA, H. D. S., BARREIRA, R. G., AND SILVA, E. M. D. Desenvolvimento de um web crawler para indexação de documentos científicos. In *Encontro de Computação e Informática do Tocantins* (2011). http://www.bandalerda.com.br/wp-content/uploads/2011/11/Desenvolvimento_de_um_Web_Crawler_para_indexacao_de_documentos_cientificos.pdf.

[19] MUSTHALER, L. A holistic approach to combating advanced persistent threats. *Network World* (November 2013). http://www.networkworld.com/article/2172114/compliance/a-holistic-approach-to-combating-advanced-persistent-threats.html.

[20] OLSTON, C., AND NAJORK, M. Web crawling. *Found. Trends Inf. Retr. 4*, 3 (Mar. 2010), 175–246. http://infolab.stanford.edu/~olston/publications/crawling_survey.pdf.

[21] RAJARAMAN, A., AND ULLMAN, J. D. *Mining of Massive Datasets*. Cambridge University Press, New York, NY, USA, 2011. http://infolab.stanford.edu/~ullman/mmds/book.pdf.

[22] RANUM, M. J. False positives: A user's guide to making sense of ids alarms. *ICSA Labs IDSC* (February 2003). http://bandwidthco.com/whitepapers/compforensics/ids/False%20Positives%20A%20Users%20Guide%20To%20IDS%20Alarms.pdf.

[23] SALTON, G., WONG, A., AND YANG, C. S. A vector space model for automatic indexing. *Commun. ACM 18*, 11 (Nov. 1975), 613–620.

[24] SPOERRI, A. Infocrystal: A visual tool for information retrieval & management. In *Proceedings of the Second International Conference on Information and Knowledge Management* (New York, NY, USA, 1993), CIKM '93, ACM, pp. 11–20. http://comminfo.rutgers.edu/~aspoerri/InfoCrystal/InfoCrystal.htm.

[25] SYMANTEC. Istr 20 - internet security threat report. https://www4.symantec.com/mktginfo/whitepaper/ISTR/21347932_GA-internet-security-threat-report-volume-20-2015-social_v2.pdf.

[26] TAN, K. Apache solr vs elasticsearch - the feature smackdown!, 2014. Retrieved: November 20, 2014, from http://solr-vs-elasticsearch.com/.

[27] THOR. Introduction: Vector space model, 1999. http://cogsys.imm.dtu.dk/thor/projects/multimedia/textmining/node5.html.

[28] TURTLE, H., AND CROFT, W. B. Evaluation of an inference network-based retrieval model. *ACM Trans. Inf. Syst. 9*, 3 (July 1991), 187–222. http://doi.acm.org/10.1145/125187.125188.

[29] UDAPURE, T. V., KALE, R. D., AND DHARMIK, R. C. Study of web crawler and its different types. *IOSR Journal of Computer Engineering (IOSR-JCE) 16* (Feb. 2014), 01–05. http://www.iosrjournals.org/iosr-jce/papers/Vol16-issue1/Version-6/A016160105.pdf.

[30] VERÍSSIMO, P., AND RODRIGUES, L. *Distributed Systems for System Architects*. Springer Publishing Company, Incorporated, 2012.

[31] W3C. Use international date format (iso). Retrieved: April 24, 2015 from http://www.w3.org/QA/Tips/iso-date.

[32] WRIGHT, K. D. Manifoldcf in action, September 2012. http://www.manning.com/wright/ManifoldCFinAction_manuscript.pdf.

Appendix A

Appendix

A.1 Implemented Ruby Code

Listing A.1: QuerySearchRank.rb

```
def initialize
 @nif          = NIFQuery.new
 @niss         = NISSQuery.new
 @phone        = PhoneQuery.new
 @dates        = DateQuery.new
 @name         = NameQuery.new
 @ip           = IPQuery.new
 @email        = EmailQuery.new
 @marketing    = MarketingQuery.new
 @password     = PasswordQuery.new
 @classifier   = Classifier.new
 @hidraES      = HidraElasticsearch.new
 @hidra        = HidraConnection.new
 @special      = #latin characters
 @dataHash     = Hash.new
 @results      = Hash.new
 @passwordList = Hash.new
 indicesData   = File.readlines("./database/parameters/commonParams.txt")
 @lastIndex    = indicesData[2].split(' = ')[1].chomp
 @ES_Fields    = ["name","date","phone","marketing","ip","nif","niss","password","email"]
 @toIndex      = nil
end

# validates and classifies the type of the data
def validate(fileID, line, option, words, lines, digest, toIndex)
 @toIndex = toIndex

 if fileID.match(/file:\/\/\/\/\//)
  filename = fileID.dup.split("file:////")[1].split('/')[-1]
 elsif fileID.match(/http[s]?:\/\//)
  filename = fileID.dup.split(/http[s]?:\/\//)[1].split('/')[-1]
 end

 if toIndex.include? "password"
  @results[:password] = Hash.new
```

```
    passHash = @password.validate(line.dup,filename)
    unless passHash.empty?
     passHash.each { |key,array|
      subtype = key.to_s
      array.each { |pair|
       username = pair[0]
       password = pair[-1]
       @passwordList.key?(:"#{fileID}") ? @passwordList[:"#{fileID}"] << password
        : @passwordList[:"#{fileID}"] = [password]
       createHash("password",subtype,password,fileID,words,lines,digest)
       if @results[:password].key?(:"#{password}")
        @results[:password][:"#{password}"] << subtype
       else
        @results[:password][:"#{password}"] = [subtype]
       end
      }
     }
    end
   end

   subvalidate("phone",line,fileID,words,lines,digest)
   subvalidate("nif",line,fileID,words,lines,digest)
   subvalidate("name",line,fileID,words,lines,digest)
   subvalidate("email",line,fileID,words,lines,digest)
   subvalidate("ip",line,fileID,words,lines,digest)
   subvalidate("marketing",line,fileID,words,lines,digest)
   subvalidate("date",line,fileID,words,lines,digest)

   return "" if option.eql?("all")
   return @results
  end

  def subvalidate(type,line,fileID,words,lines,digest)
   @results[:"#{type}"] = Hash.new

   if @toIndex.include? type
    typeClass = instance_variable_get("@#{type}")
    item = typeClass.validate(line)
    if item
     item.each { |i|
      subtype = typeClass.getSubtype(i)
      createHash(type,subtype,i.to_s,fileID,words,lines,digest)
      @results[:"#{type}"][:"#{i}"] = [subtype]
     }
    end
   end
  end
```

Listing A.2: PhoneQuery.rb

```
#!/usr/bin/ruby

class PhoneQuery

  def validate(string)
```

```
    Dir['./database/list/phone/*'].empty?
     ? content = nil
     : content=$aux.folderReader("./database/list/phone")

    array = []

    if content == nil
    # REGULAR EXPRESSIONS VALIDATION
      s = string.scan(/(?:|\t|[a-zA-Z]|^)(?:\+?0?0?351)?
       ((?:(?:2[0-9]{8})|(?:9[1-4,6][0-9]{7})))
       (?:\\|\s|\t|[a-zA-Z]|$)/)
      s.each { |m,n|
        array << m.to_s if m.to_s!=""
      }
    else
    # LIST OF PHONE VALIDATION
      content.lines.each { |line|
        l = /\b#{line.chomp}\b/i.match(s)
        array << l.to_s.strip if l
      }
    end

    return array unless array.empty?
  end

  def getSubtype(param)
    param = param.to_s if param.class != String

    if param[0].eql?("9")
      cells = $aux.fileReader("./database/phones/cell-phones.txt")
      hash = phonesToHash(cells)

      hash.each { |key,value|
       if key.size == 2
        return value if param[0..1].eql?(key)
       elsif key.size == 3
        return value if param[0..2].eql?(key)
       end
      }
    end
    return "phone"
  end

  def phonesToHash(raw)
    phoneHash = Hash.new { |hash, key| }
    raw.each_line do |line|
      phoneHash[line.split('=')[0].chomp] = line.to_s.split('=')[1].chomp
    end
    return phoneHash
  end

end
```